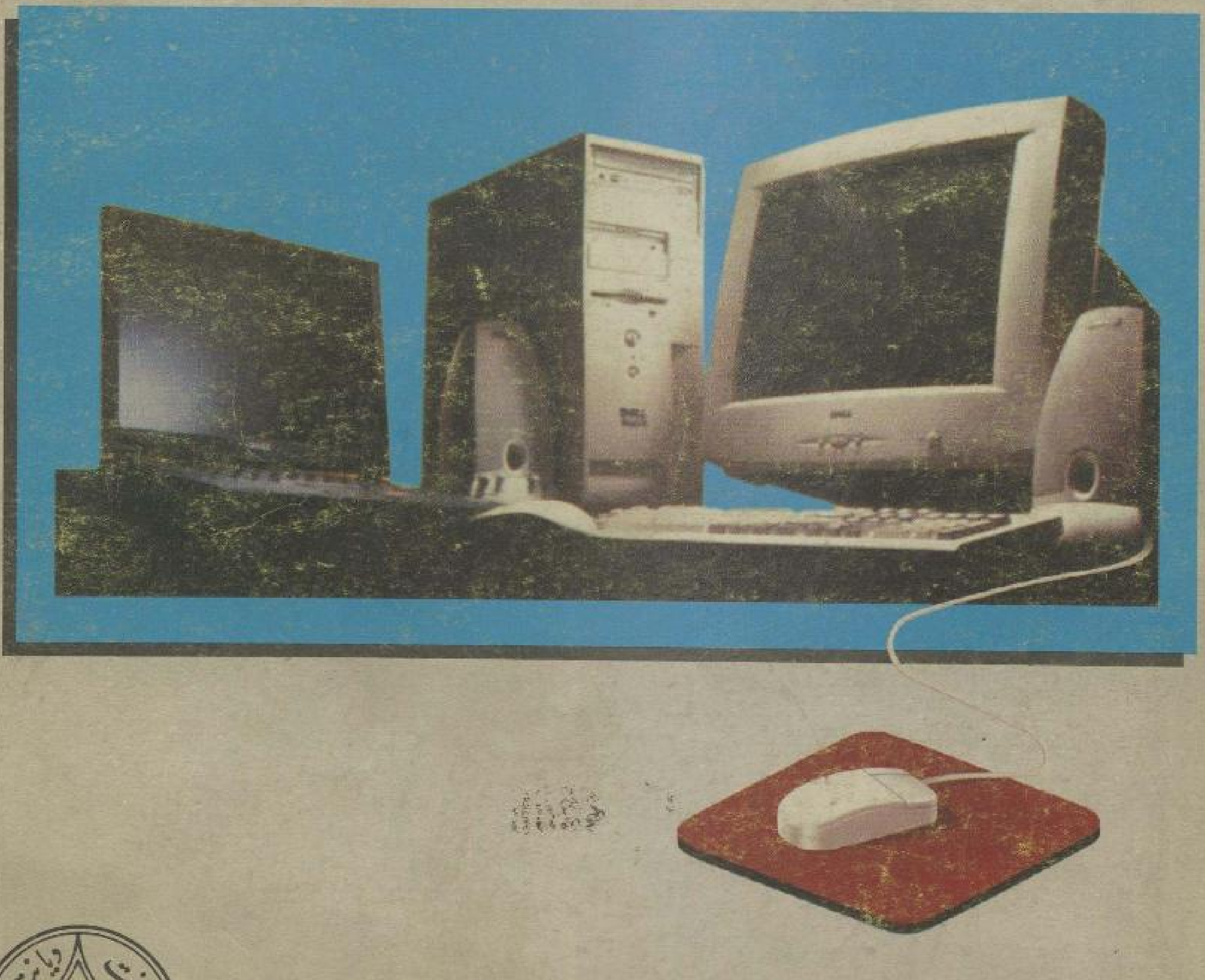
Computer Science

For Class XII



Punjab Textbook Board, Lahore.

Appeal

Dear Students

Punjab Textbook Board is your own organization. It produces quality and cost effective textbooks for you. These textbooks are produced under the supervision of experienced educationists. They are developed to enhance your creative faculties.

Unfortunately, some private publishers produce Guide-books, class-notes, test-papers, and similar kinds of books which contain sub-standard textual material. The children may cram this material and get-through their school examination, but this kind of material does not help creative capabilities.

Parents, teachers and students are not bound to buy these un-authorized books.

If somebody compels you to buy any of these books, please inform in writing to the Chairperson, Punjab Textbook Board.

Dr. Fouzia Saleemi
Ph.D Physics (Glasgow)
(Sitara-e-Imtiaz & Izaz-e-Fazeelat)
Chairperson
Punjab Textbook Board
21-E-II, Gulberg-III, Lahore.

Computer Science

ICS Text Book for Class XII

Curriculum Development & Text Book Production Wing • H-9 Islamabad •
272
LIBRARY

Punjab Textbook Board, Lahore.

All rights reserved with the
Punjab Textbook Board, Lahore

Compiled by:

Punjab Information Technology Board, Lahore

Authors:

Dr. Hamayun M. Mian
Omar Khan
Atif Alvi

Supervision:

Mazhar Hayat

Senior Artist:

Zaheer-ul-Haq

Director(Tech.):

Prof. Naseer-ud-Din

Publishers:

Nadeem Brothers Lahore.

Printed at:

Artistic Printers Lahore.

For Comments and Suggestions:

- Punjab Information Technology Board
  ics@pitb.gov.pk
- Punjab Textbook Board, Lahore.

PREFACE

Contents of this book have been designed specifically to cover the course material prescribed for class 12 within the ICS curriculum. Main emphasis has been on the simplicity and friendliness of the contents and language. No attempt has been made to cover the subject matter comprehensively, as this is not the main purpose of the ICS programme. Thus the book concentrates on introducing the essentials of relational databases and computer programming. It is hoped that if students become familiar and comfortable with these fundamental concepts, they can quickly expand their knowledge to any degree of expertise by using several on-line interactive teaching tools available e.g. Step-by-Step© and the MSDN©. It is assumed that the computer labs will be used extensively to give students practical experience of concepts covered in this book.

The Punjab Textbook Board also plans to publish a hypertext version of this book in a CD form. This CD will also contain useful exercises and examples to cover the Practical component of the ICS curriculum.

ACKNOWLEDGEMENTS

This book is a result of a collaborative effort involving several young computer professionals. Particular acknowledgements are due to the following individuals for their hard work and contributions to the project:

Atif Alvi.
Omar Khan.
Abu Zar.

Dr. Hamayun M. Mian
October 2001

CONTENTS

Part I: Introduction to Databases

CHAPTER 1

INTRODUCTION TO DATABASES 9

Overview 9
1.1 History and Basic Concepts 9
1.2 Why Database Systems? 10
Exercises 11

CHAPTER 2

DATABASE CONCEPTS AND TERMINOLOGY 12

Overview 12
2.1 Understanding a Database System 12
  2.1.1 Database Program or DBMS 13
  2.1.2 Database Access Language 13
  2.1.3 Users 14
  2.1.4 Integrated Development Environment (IDE) 15
  2.1.5 Database Objects 16
Exercises 18

CHAPTER 3

TABLE CREATION IN ACCESS 19

Overview 19
3.1 Concept of a Table 19
3.2 Creating Tables in Access 21
  3.2.1 Setting up a database in Access 21
  3.2.2 Creating a table structure 24
  3.2.3 Setting the primary key 26
  3.2.4 Saving a Table Structure 28
Exercises 29

CHAPTER 4

THE ACCESS IDE 30

4.1 Opening a Table 30
4.2 Adding Data to a Table 31
  4.2.1 Tips for Adding Data 32
4.3 Deleting data from a Table 33
4.4 Sorting Data 33
4.5 Copying and Moving Data 35
4.6 Locating and Replacing Data 35
  4.6.1 Finding Data 35
  4.6.2 Replacing Data 37
4.7 Formatting a Table 37

4.7.1 Formatting Data *38*
4.7.2 Arranging Table Columns *39*
4.8 CHANGING TABLE DESIGN 40
4.8.1 Switching between Design view and Datasheet view *40*
4.8.2 Changing a field name *41*
4.8.3 Inserting a new Column / Field *41*
4.8.4 Deleting a Column *42*
EXERCISES 42

CHAPTER 5

DATABASE DESIGN 44
5.1 CONCEPT OF A RELATIONAL DATABASE 44
5.2 DATABASE DESIGN PRINCIPLES 46
5.2.1 Use multiple tables *46*
5.2.2 Achieve flexibility by dividing data into small units. *46*
5.2.3 Choose the right Data type for fields *47*
5.2.4 Set a Primary key *48*
5.2.5 Normalize Tables *48*
EXERCISES 50

CHAPTER 6

RELATIONSHIPS 52
6.1 ENTITY – RELATIONSHIP MODEL 52
6.1.1 Entity *52*
6.1.2 Relationship *52*
6.2 ADVANTAGES OF DEFINING RELATIONSHIPS 53
6.3 DEFINING RELATIONSHIPS IN ACCESS 54

CHAPTER 7

CREATING SIMPLE QUERIES 57
7.1 CREATING QUERIES FROM SCRATCH 57
7.2 QUERIES BASED ON CRITERIA 60
7.3 WILDCARDS 61
7.4 JOINS : QUERYING MULTIPLE TABLES 61

CHAPTER 8

CREATING AND FORMATTING FORMS 64
8.1 CREATING A SINGLE TABLE FORM 64
8.2 EDITING RECORDS THROUGH FORMS 66
8.3 ADDING NEW RECORDS THROUGH FORMS 66
8.4 CREATING A FORM FROM TWO TABLES 67
8.5 EDITING AND ADDING DATA IN MULTI-TABLE FORMS 69
EXERCISES 69

CHAPTER 9

CREATING AND FORMATTING REPORTS **70**

9.1 Creating a Single Table Report 70
9.2 Creating a Report from two tables 73
Exercises: 75

CHAPTER 10

SQL **76**

Introduction 76
10.1 The SQL Interface 77
10.2 The CREATE Statement 77
10.3 Adding Data with the INSERT statement 79
10.4 Retrieving Data with the Select Statement 79
10.5 Modifying Data with the UPDATE Statement 81
10.6 The Delete and Drop Table Statements 81
10.7 CODD's 12 RULES 82
  10.7.1 THE 12 RULES *82*

Part II: Programming in C

CHAPTER 11

INTRODUCTION TO COMPUTER SCIENCE **85**

11.1 Computer organization 85
  11.1.2 System Buses *86*
  11.1.3 The Central Processing Unit (CPU) *86*
  11.1.4 Main Memory *88*
  11.1.5 Secondary Memory *88*
  11.1.6 Input/Output Devices *89*
  11.1.7 I/O Ports *89*
11.2 Instruction Execution Cycle 89
11.3 Programming Languages 90
  11.3.1 Machine Language *90*
  11.3.2 Assembly Language *91*
  11.3.3 Higher-Level Languages *91*
11.4 Characteristics of Higher-Level Languages 92

CHAPTER 12

BASIC STRUCTURE OF A PROGRAM **94**

12.1 Our First C Program 94
Exercises 96

CHAPTER 13

LEXICAL ELEMENTS OF C **99**

13.1 The Language 99

13.2 Alphabet of the C language 101
  13.2.1 Keywords *102*
  13.2.2 Variables / Identifiers *103*
  13.2.3 Constants *103*
  13.2.4 String Constants *103*
  13.2.5 Operators and Punctuators *103*
13.3 Comments 104
Exercises 104

CHAPTER 14

DATA TYPES 106
14.1 Declarations 106
14.2 The Data Type char 106
14.3 The Data type int 108
14.4 The Data type long 109
14.5 Data types float and double 109
14.6 Type Qualifier const 110
Exercises 110

CHAPTER 15

OPERATORS 113
15.1 Arithmetic Operators 113
15.2 Increment and Decrement Operators 114
15.3 The Assignment operator 115
15.4 Relational Operators 116
15.5 Logical Operators 117
15.6 Operator precedence and associativity 118
Exercises 119

CHAPTER 16

INPUT / OUTPUT 120
16.1 printf ( ) the output function 120
16.2 scanf ( ) the input function 121
Exercises 123

CHAPTER 17

DECISION STATEMENTS 124
17.1 The if Statement 124
17.2 The if else Statement 125
17.3 Dangling else 126
17.4 The Conditional Operator 126
17.5 The switch Statement 127
Exercises 128

CHAPTER 18

LOOPS **129**

18.1 THE WHILE STATEMENT 129
18.2 THE FOR STATEMENT 130
18.3 THE DO STATEMENT 131
EXERCISES 132

CHAPTER 19

FUNCTIONS **135**

19.1 IMPORTANCE OF FUNCTIONS 135
19.2 FUNCTION DEFINITION 137
19.3 FUNCTION CALL PROCESS 137
19.4 FUNCTION DECLARATIONS 138
19.5 ADVANTAGES OF USING FUNCTIONS 138
19.6 C PROGRAM STRUCTURE AND THE `MAIN ( )` FUNCTION 139
EXERCISES 139

Chapter 1

Introduction to Databases

Overview

In the past two decades Database Management Systems have evolved from being obscure computer applications to being an integral part of the computing environments of most organizations. The availability of databases on personal computer has even made them popular in small businesses and households. This widespread use has made the basic working knowledge of databases an essential part of any computer science curriculum. This chapter presents a short history of how databases have evolved and the advantages that are to be gained from their use.

1.1 History and Basic Concepts

University of Pennsylvania built the first all electronic computer in late 1945 and named it **Electronic Numerical Integrator and Calculator** (ENIAC). It remains the largest computing machine ever built, as far as physical size is concerned. However, it could only store a maximum of 20 numbers at one time. Compare this with a typical modern personal computer that can easily store hundreds of millions of numbers. Since its invention, the electronic computer has doubled in storage capacity and computational capacity every 18 months. It is believed that this rate of growth will continue for many years to come. This is commonly known as **Moore's Law**. The incredible growth in the capacity of computer to store and process data has allowed computer scientists to develop various techniques to record large amounts of data and to retrieve required information from it. Such a collection of data is called a ***database*** and the techniques to manage and search a database are called ***database management systems***.

Data: We can think of ***data*** as any information that we want to store. Computers store information as a sequence of binary digits (e.g. 0101100001). Therefore data for computers is any information convertible to a digital form. During the earlier period of computing, computer data consisted only of text and numbers. With improvements in storing capacity of computers it has now also become possible to save information like sound, pictures etc.

Computer database management systems are efficient computer programs that emulate the databases we find in our daily experience. For example, a paper telephone directory is an example of a non-electronic database. We can store the entire contents of a city's telephone directory in a computerized telephone directory database system. We can then use this database with appropriate search programs to find a required telephone number in a fraction of a second. The computerized search process is thus many times faster and efficient than the normal manual look-up or search method. The speed and efficiency of the computer allows us to do a job in a fraction of the time required to do it manually.

More recently, the Internet has added to the usefulness of an electronic database by providing a way for thousands of users to get information from the same database simultaneously. No wonder that database applications are among the most commonly used computer programs.

Database: A collection of data containing information about a particular topic stored in an efficient and compact manner.

Database Management System (DBMS): A set of computer programs (software) to build, maintain and access the database in a convenient and efficient manner.

In the above definitions by **efficient** we mean that it is possible to look up the required data quickly, and by **compact** we mean that the stored data takes up as little space as possible. Some of the popular database management systems are Access, Oracle, MySQL and Sybase.

1.2 Why Database Systems?

Suppose we are given the task of storing information about all the students of a school. We are required to store each student's name, address, age, class, subjects, fee payment status, games etc. for the school record. Let us look at the advantages of storing this information in a database system by comparing it with a system that stores the same data about each student in various unrelated files on a computer or otherwise. The major advantages of the database system are:

- **Information is recorded consistently and with minimum repetitions.**
  Mostly, all the information about a student is recorded only once in a database. For example, suppose a student is in both the cricket and hockey teams. The student's name will be entered only once in the database and if queried, the database will correctly generate the lists of players of the cricket and hockey teams, including the student's name in both the teams. On the other hand, an ordinary file system will have a separate file for each team forcing us to store the student's name twice on the computer. This wastes space, effort and time. It is also possible by mistake to spell a student's name differently in the two separate files.

- **Data can be accessed easily and efficiently.**
  A database system allows information to be retrieved conveniently. For example it will be easy to generate a list of all the students who have taken a certain subject. A filing system needs a separate special program to handle such an inquiry. Also, such information retrieval is much faster with a database than with an ordinary filing system.

- **Data can be checked for various conditions.**
  It is easy in a school database system to enforce the condition that a student should not be allowed to attend school if he or she has not paid the school fee. Moreover, it would also be a simple task to introduce other such conditions. A

student's data can then be automatically checked to make sure that it satisfies these conditions. These are called ***consistency constraints*** in database terminology. In a filing system, it is difficult to apply such conditions, if they involve data from different files, and to introduce any such new conditions.

- **Data is either entered as a whole or it is not entered at all.**
  Assume that we have to record the fact that a student has paid the school fee. For this, we need to add the student's name to the list of students who have paid their fee and to remove it from the list of those who have not. Consider the situation in the file processing system, where we have successfully entered the student's name to the list of students who have paid, but have not yet removed his or her name from the other list. Now suppose that at this point in time the power fails and we forget to update the second list. We will then have an inconsistent set of files with the student's name on both the lists. This is called the ***data atomicity problem***, in database terminology. In a database system, such a situation will not occur. The database will either enter the student's name in the first list and remove it from the second list, or it will perform no action. That is, it would make sure that such a compound action either takes place as a whole or it does not take place at all.

Exercises

1. Do you know of any large databases that have been developed around the world or in Pakistan?

2. Can you give examples of information that have become popular in a digital format in the past 5 years?

3. Think of a task that will become easier to manage if it is done using an electronic database system. List the disadvantages of doing it manually.

4. Define the following terms:
   (a) *Data* (b) *Database management system*
   (c) *Consistency constraints* (d) *Data atomicity*

5. What are the advantages of using a database system?

6. Can you think of any disadvantages of using a database system?

Chapter 2

Database Concepts and Terminology

Overview

This chapter provides an introduction to the concepts that are common to most of the popular Database Management Systems available today. No specific Database Management System will be discussed here. The theory presented in this chapter provides a background for the more practical work we will perform in the following chapters. You should become comfortable with the material presented in this chapter, as it will be used extensively in later chapters.

2.1 Understanding a Database System

To better understand a database system environment we may think of it as consisting of five components: ***database program***, ***database access language***, ***users***, ***Integrated Development Environment (IDE)*** and ***database objects***. Let us now have a closer look at each of these components.

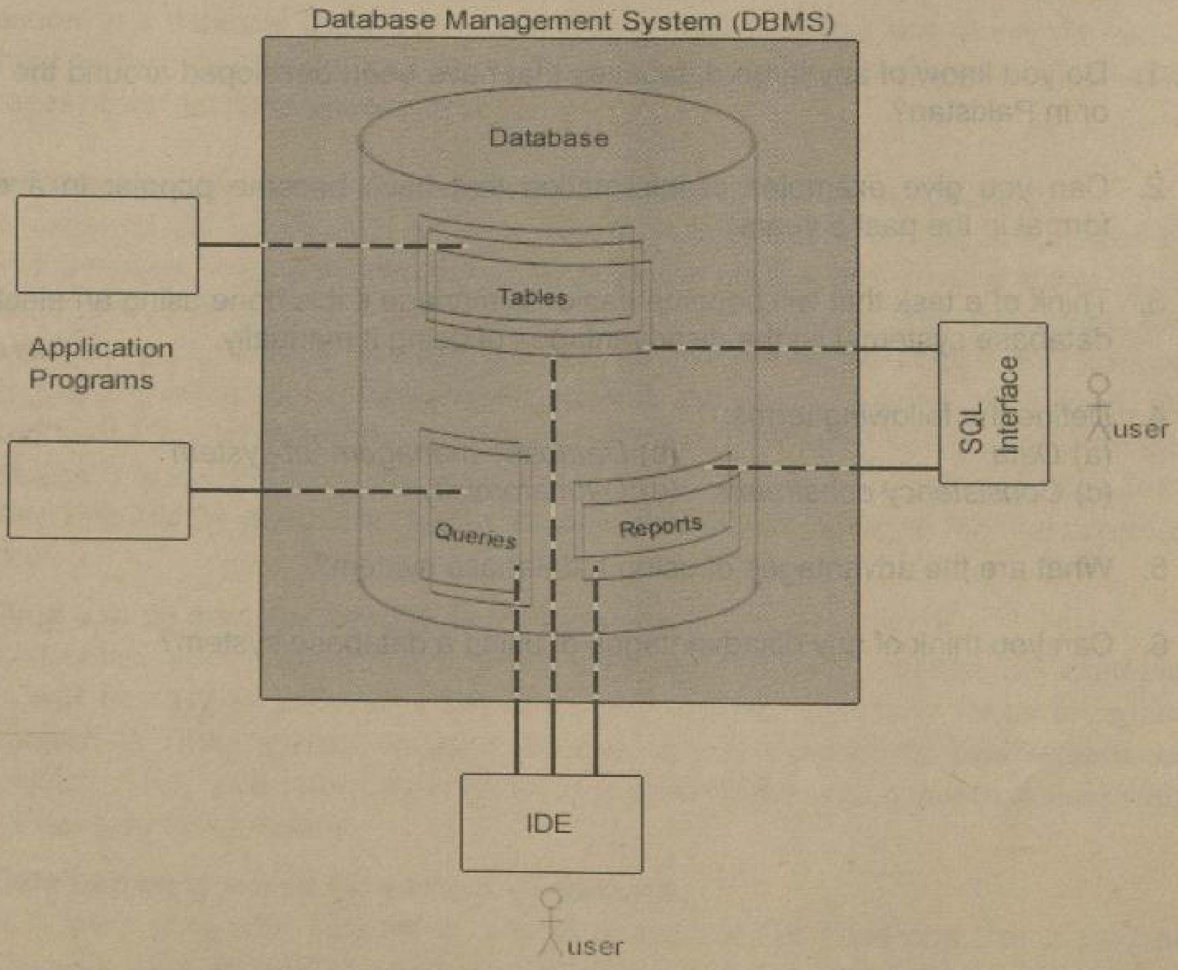

Database Management System (DBMS)
Database
Tables
Application
Programs
Queries
Reports
SQL
Interface
user
IDE
user

Figure 2.1: Various Components of a DBMS environment

2.1.1 Database Program or DBMS

This is the software that actually stores the input data on the computer using different algorithms. The database program also handles various requests for data retrieval. Another name for it is ***Database Management System*** or ***DBMS***. Normally a user can use a DBMS in at least three different ways as shown in fig. 2.1:

1. The user writes a set of appropriate commands in a Database Access Language, e.g. SQL and submits these to the DBMS. The DBMS translates the user commands into a form that is suitable for communicating with the database and submits it to a specific part of the DBMS called the ***database engine***. The database engine generates a set of results desired by the user, compiles these into a user readable form, called an ***inquiry report***, and then displays them on the user's screen.
2. Alternately, where the user does not or cannot use a formal inquiry language such as SQL, the same process may be performed through a user-friendly graphical facility called an Integrated Development Environment (IDE), as explained in section 2.1.4. In this method the user specifies his/her inquiry by filling in a visual or virtual form, presented on the screen. This Form is submitted to the DBMS that returns the inquiry results to the user's screen or to a printer. The DBMS that you will study this course, called Microsoft Access, uses this method.
3. Quite often the results of a particular inquiry are not required for direct viewing or printing by a user. Instead the results are used within a general-purpose program, called an ***application program***. In such cases most DBMS provide a facility where a computer program can send an inquiry directly to the database and the results are also communicated back directly to the program. Such inquiries are called ***imbedded inquiries***. Microsoft Access and most other popular DBMS provide facilities for imbedding inquiries inside user application programs.

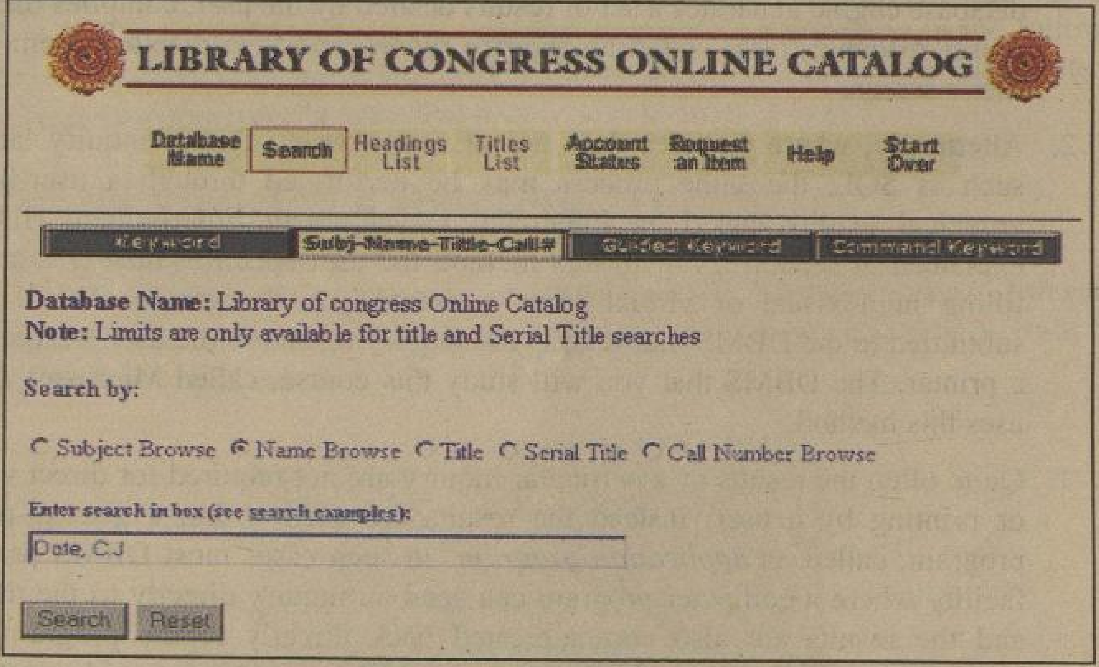
2.1.2 Database Access Language

As mentioned earlier, many types of databases can be accessed through an access or query language. **Structured query language (SQL)** is the most common query language nowadays. Relational databases are required to have a query language. A set-oriented database access language is what differentiates a ***Relational Database Management System (RDBMS)*** from a Database Management System (DBMS). Most RDBMSs today implement SQL as their set-oriented database language. Database administrators use query language to create and maintain databases, and users use query language to get data from the database and to make changes to the stored data. SQL is a very powerful concept and is supported by most relational database systems. Since Access does not allow users to interact directly with the database through SQL, it does not form a part of your course. Access however does implement SQL and uses it in various processes hidden from the user. For those of you interested in SQL we have provided a complete chapter on SQL (chapter 10).

2.1.3 Users

There are three kinds of people who use a database system: ***end users***, ***database administrators (DBAs)***, and ***application programmers***.

- **End user:**
  End user is a person who uses a database application without knowing anything about databases. The end users interact with the database through an interface without writing any programs themselves. For example the Figure 2.2 shows the interface of the library of Congress online catalog.


LIBRARY OF CONGRESS ONLINE CATALOG
Database Name
Search
Headings List
Titles List
Account Status
Request an Item
Help
Start Over
Keyword
Subj-Name-Title-Call#
Guided Keyword
Command Keyword
Search
Reset

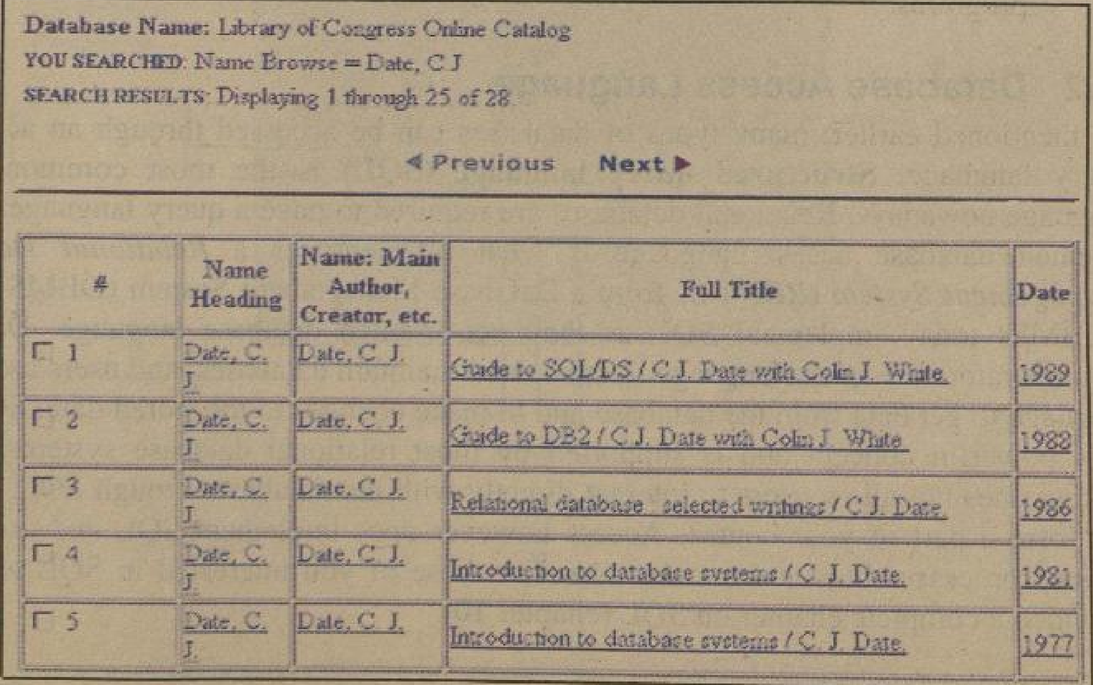
Database Name: Library of congress Online Catalog
Note: Limits are only available for title and Serial Title searches

Search by:

○ Subject Browse ● Name Browse ○ Title ○ Serial Title ○ Call Number Browse

Enter search in box (see search examples):
Date, CJ

Figure 2.2: Example of an end user interface


◀ Previous
Next ▶

Database Name: Library of Congress Online Catalog
YOU SEARCHED: Name Browse = Date, C J
SEARCH RESULTS: Displaying 1 through 25 of 28

#	Name Heading	Name: Main Author, Creator, etc.	Full Title	Date
☐ 1	Date, C. J.	Date, C J.	Guide to SQL/DS / C J. Date with Colin J. White.	1989
☐ 2	Date, C. J.	Date, C J.	Guide to DB2 / C J. Date with Colin J. White.	1988
☐ 3	Date, C. J.	Date, C J.	Relational database : selected writings / C J. Date.	1986
☐ 4	Date, C. J.	Date, C J.	Introduction to database systems / C J. Date.	1981
☐ 5	Date, C. J.	Date, C J.	Introduction to database systems / C. J. Date.	1977

Figure 2.3: Results of an online query to a database

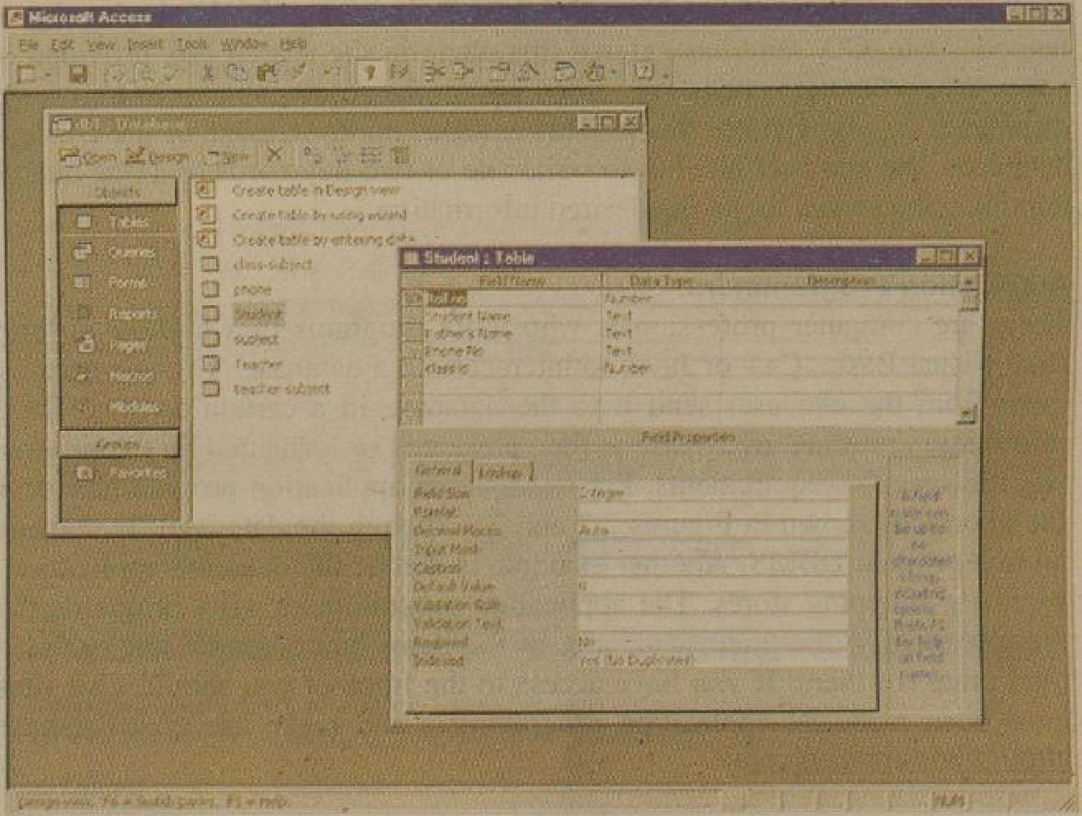
This system is used to find out what books are present in the library. In the screen shot shown in Figure 2.2 above the user is searching for books by the author named 'Date, C.J.' and has entered this name in the search box provided on the page. When the user clicks the search button the results shown in Figure 2.3 are displayed. As you can see this process requires no knowledge of the concept of a database. The end user just has to write one or two keywords in a box and click a button on the screen to get the desired information.

- **Application Programmers:**
  These are computer professionals who write programs in a high-level language (like Visual Basic, C++ or Java) to interact with a database. These programs take input from the end user send it to the database in a certain format. The basic information returned by DBMS is then presented or formatted by the programs to suit the specific requirements. For example an application programmer designed the interfaces shown in Figures 2.2 and 2.3, though the data itself is managed by an independent DBMS. Another example is that of the database application used by many shopping stores. The application programmers write programs that get the prices, from the prices database, of the items being bought and calculate the total price for them. If you have access to the Internet you should visit sites that have been made by application programmers and make use of a database e.g. http://www.amazon.com

- **Database Administrator (DBA):**
  In a ***multi-user environment***, where more than one person use the same database system at the same time, a dedicated person is usually provided to install the actual database software and then manage its day to day working or operation. Such a person is called a ***database administrator*** or a ***DBA***. A DBA works closely with database programmers in designing databases according to a design that is suitable for the organization whose data is to be managed by the DBMS. A DBA must ensure that each user is permitted to make a change only to those parts of the database for which he has authorized access. It is also a responsibility of a DBA to ensure that regular data backups are made, so that if a database is lost or corrupted due to hardware malfunction, a copy of data is available to restore the database to its previous state.

2.1.4 Integrated Development Environment (IDE)

An IDE is an interface that is used by database designers and application programmers to create databases and database applications. An IDE simplifies the tasks of creating and using a database. Instead of creating a database application manually the user just enters the specifications of the required application into a graphical interface and the IDE builds the application automatically. The IDE thus hides the complicated details from the user. A user therefore does not have to have a detailed knowledge of the concepts of database systems. Microsoft Access is an example of a database IDE. It provides the user with a graphical interface that can be used for database creation, queries, report generation and other similar tasks. Figure 2.4 shows the Access IDE environment with an open table in design view.


Microsoft Access
File Edit View Insert Tools Window Help
Open Design New
Objects
Tables
Queries
Forms
Reports
Pages
Macros
Modules
Groups
Favorites
Create table in Design view
Create table by using wizard
Create table by entering data
class-subject
phone
Student
subject
Teacher
teacher subject
Student : Table
Field Properties
General
Lookup
NUM

Field Name	Data Type	Description
[illegible]	Number	
Student Name	Text	
Father's Name	Text	
Phone No	Text	
Class Id	Number	

Field Size	Integer
Format	
Decimal Places	Auto
Input Mask	
Caption	
Default Value	0
Validation Rule	
Validation Text	
Required	No
Indexed	Yes (No Duplicates)

Figure 2.4: The Microsoft Access Integrated Development Environment (IDE)

2.1.5 Database Objects

In this course we are going to study relational databases. To make the task of understanding a database easier we will think of it as made up of various parts or objects. Here is an introduction to the database objects that we will study throughout this book.

- **Tables**

  The concept of a ***table*** is central to relational, databases. All data in a relational database is stored in tables. This is a fundamental property of a relational database. In fact this is where the term *relational* is derived from. Relation is just a mathematical term for a table (there are certain criteria that the table must have, we will discuss those in the next chapter). Figure 2.5 presents a simple table in a graphical form.

Employee #	Name	Department	Salary
E1	Aamir	Marketing	25000
E2	Shafiq	Engineering	20000
E3	Tahir	Engineering	30000
E4	Omer	Development	25000

Figure 2.5: A simple table.

As you can see, this table stores information about employees. Note the structure of the table. Information about each employee is stored in a separate row and each row is divided into fields containing different data in a consistent manner. A typical database contains lots of tables, which are identified by separate names. One of the most important tasks in designing a database is deciding which tables to include in the database and what the fields for each table must be.

- **Queries**

  If all we wanted to do was to store data, databases would be of little use. We could store the same data in simple text instead. The real advantages of a database become apparent when we want to retrieve data that fulfills a certain criteria. For example, let us suppose that a database contains a table similar to the one in Figure 2.5, but which contains information about a 10,000 employees of a corporation. We want to find out the names of employees who have a salary of Rs. 30,000 or more. Databases allow us to retrieve this kind of information is a matter of seconds through the use of ***queries***. The output of a query is also in form of a table. This is another important principle of relational databases; all operations in a relational database have tables as their inputs and outputs. For example if we ran a query to get information about employees with a salary of 25000 on the table in Figure 2.5, we would get the following output

Employee #	Name	Department
E1	Aamir	Marketing
E4	Omer	Development

Figure2.6: A simple query output.

As you can see we got a table as the output. The advantage of this is that we can now use this output as an input to another operation.

- **Forms**

  Many users may find tables rather difficult to work with. ***Forms*** are therefore provided, as an alternative, by database management systems such as Microsoft Access to make it easier to view and work with data in tables. A form on your computer can be made to look exactly like a fill-in-the-blanks paper form most companies use to gather data in every day life. The end-user of a database say, a secretary may find such a form a more familiar and friendlier format to enter or retrieve data from a database table. It is however not essential for database management systems to provide a facility for producing forms on your screen. Most commercial database management systems nevertheless provide this facility since it makes the task of the end user much more easier.

- **Reports**

  Another facility commonly provided in database management systems is called ***reports***. This facility enables the user to obtain the results of an inquiry printed in an appropriate layout or format. It must be highlighted here that the concept and facility of reports is not an integral part of a DBMS. It is merely an additional useful facility to make outputs more friendly and according to specific needs of a particular user or organization. Both the concepts of reports and forms shall be dealt with at a greater detail later in this book.

Exercises

1. List the major parts of a database management system.
2. If you have access to the Internet visit the site www.google.com. This is a search engine that allows users to locate web pages that have certain key words present in them. The information about all the web pages is stored in a huge database with a size that exceeds 1000 gigabytes.
3. Search the database with a key word such as 'DBMS'.
4. Note the time it takes the database to search for this information. (The time will be shown on top of the results). Are you surprised by the time it took the database for the search?
5. When you are using this site what kind of a user are you?
6. Can you list the two characteristics that a relational database system must possess?
7. What are the advantages of using an IDE? Can you list any disadvantages?
8. What are the database-objects used to store data and retrieve data?
9. Why are forms and reports provided by database systems?

Chapter 3

Table Creation in Access

Overview

Relational database management systems work with tables. In fact this is what makes them relational. In this chapter we take the first practical step in the creation of a database system. We will use Microsoft Access to create our first table. The Access environment may seem strange and difficult at first, but as you work through the examples it will become more familiar and easier to work with. A more thorough guide to the Access working environment is provided in the next chapter.

3.1 Concept of a Table

To explain the various concepts of a relational-database we now introduce an example database that we will keep on developing throughout the book. This database will be designed to automate the data management of a typical school.

The very first thing to remember is that that a school can be viewed as a ***system*** that consists of various types of things or components such as students, teachers, classrooms etc. These main parts or components of a system are called ***entities*** of the system. Thus student, classroom, teacher etc. are all examples of entities of a system called a school.

In a manual record keeping system for a school information on each entity of the system will be kept together as one ***aggregation***. Such aggregations of information will often be called **Registers** or **Books**. Thus we expect to see student registers and teacher registers inside a school. The same in a relational-database system will be called ***tables***. Thus all records pertaining to students of a school will be maintained in a *table* that will most probably be called the 'student-table'. Similarly, there will be a teacher-table and a class-table in such a database.

As a first task, let us see how information about each student is represented in a relational-database. Data about a student shall typically include information such as:

Roll no
Student Name
Father's Name
Date of birth
Phone no

Let us assume that at present the school has a register containing this information in the form shown in Figure 3.1:

Roll no	Student Name	Father's Name	Date of birth	Phone no
1234	Abdul Hameed	Amir Hameed	3/19/90	586-9830
1235	Usman Khan	Tariq Khan	2/25/91	575-1312
1236	Irfan Chaudhry	Asim Chaudhry	9/4/90	767-1247
1237	Abdul Hameed	Shahid Hameed	4/2/90	555-5590

Figure 3.1: Typical school register data in table form

The above form or format for representing data is called a ***table*** in relational-database terminology. One must notice that data about each student is stored in a separate row. For example the row containing information on a student 'Abdul Hameed' with Roll no 1234 is:

1234	Abdul Hameed	Amir Hameed	3/19/90	586-9830

Figure 3.2: Table row for Roll no. 1234

Whereas there is another student in our table with same name i.e. 'Abdul Hameed', whose roll no is 1237. Information on him is placed in a separate row:

1237	Abdul Hameed	Shahid Hameed	4/2/90	555-5590

Figure 3.3: Table row for Roll no. 1237

In relational-database terminology each row of data is called a ***record***. Another important thing to notice is that each row is further divided into columns where each column represents a particular piece of information (also called ***attribute*** or ***property***) about the student. In our example, the first column of the row contains the roll number of the student. Similarly, the second column, shown in Figure 3.4, of our table contains the name of the student.

Abdul Hameed
Usman Khan
Irfan Chaudhry
Abdul Hameed

Figure 3.4: The Name column

In relational-database terminology, each member of a column is termed a ***field***. Thus, 'Irfan Chaudhry' is a specific field in the column *Name*.

Table
Form of data representation in databases in which all the items of information are shown in rows and columns.

Record
Each **row** of a table representing a set of information about, say a student.

Column
Each **column** of a table provides the same piece of information (also called ***attribute*** or ***property***) for each record in a table. For example our particular table has five columns called (i) Roll no, (ii) Name, (iii) Father's Name, (iv) Date of Birth, and (v) Phone no.

Field
Each individual member of a column of a table e.g. 'Irfan Chaudhry' in the 'Name' column of our table.

3.2 Creating Tables in Access

In a relational-database all information is contained as values in tables. A database management system essentially consists of facilities for declaring databases, creating tables in them, maintaining changes in these tables and for enquiries on the contents of tables. In Access a database contains tables plus all the objects used to manage the data in these tables. These objects provide facilities for:

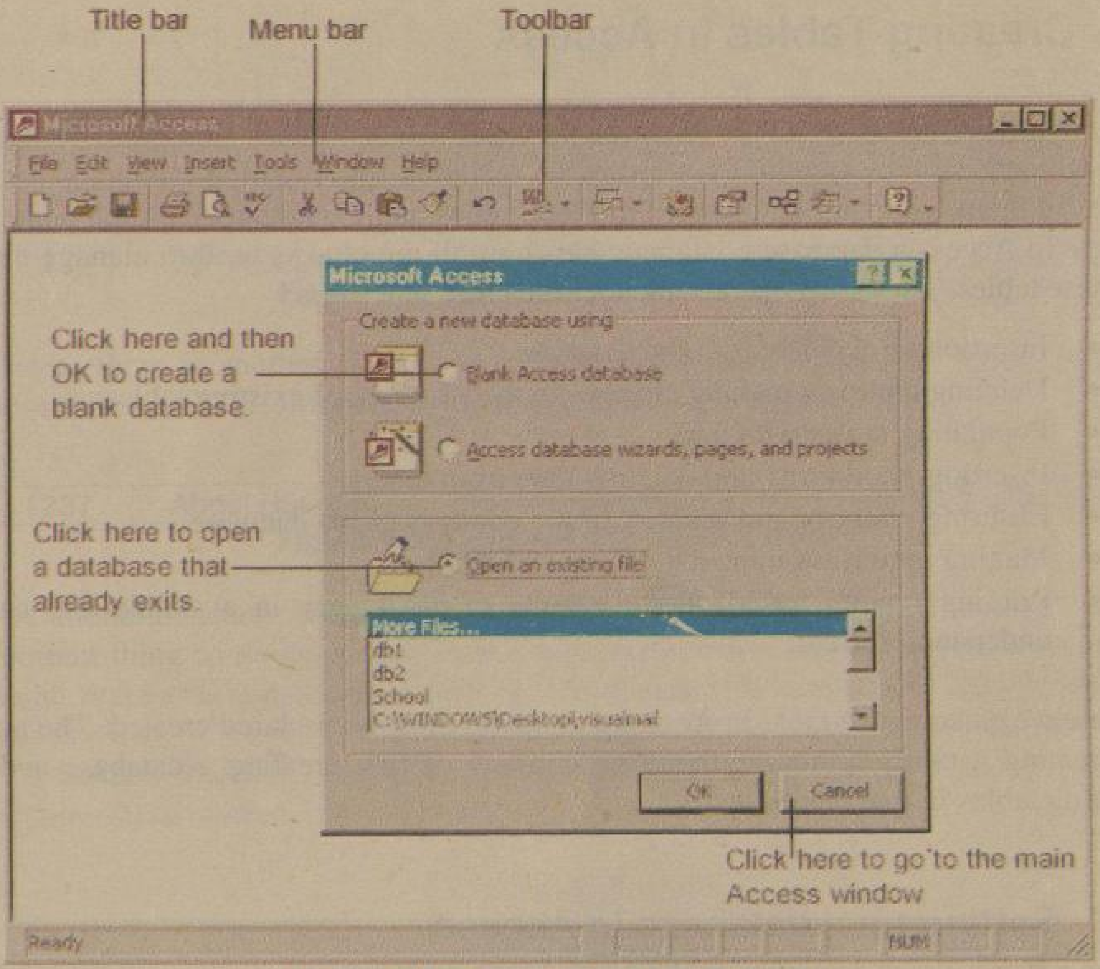
- Insertion of new tables in the database
- Deleting tables or making changes to the structure of existing tables.
- Populating tables with data.
- Inserting, modifying and deleting data from tables.
- Ensuring integrity and security of the contents of the database.
- Making inquiries on the data contained in tables.
- Printing reports based on the results of an inquiry in a simple and easy to understand format.

Before we can create a table in Access a database must be declared/created. The process of creating a table in Access therefore consists of first creating a database and then creating tables in that database.

3.2.1 Setting up a database in Access

When setting up a database in Access we have a choice between creating a blank database or a database that has some tables already defined in it. Since we want to learn how to create tables for ourselves we will start by setting up a blank database and then create tables in it ourselves.
When you start Access it presents you with the dialog box shown in Figure 3.5:
You have four options at this point.

1. **Create a new database from the dialog box**
   To do this, click on the **Blank Access database** option and then click **OK**.
2. **Create a new database from the main menu**
   To do this, click the **Cancel** button. In the Access main menu click on **File** and then choose **New** or press the **Ctrl** and **N** keys simultaneously. The dialog box shown in Figure 3.6 will appear:

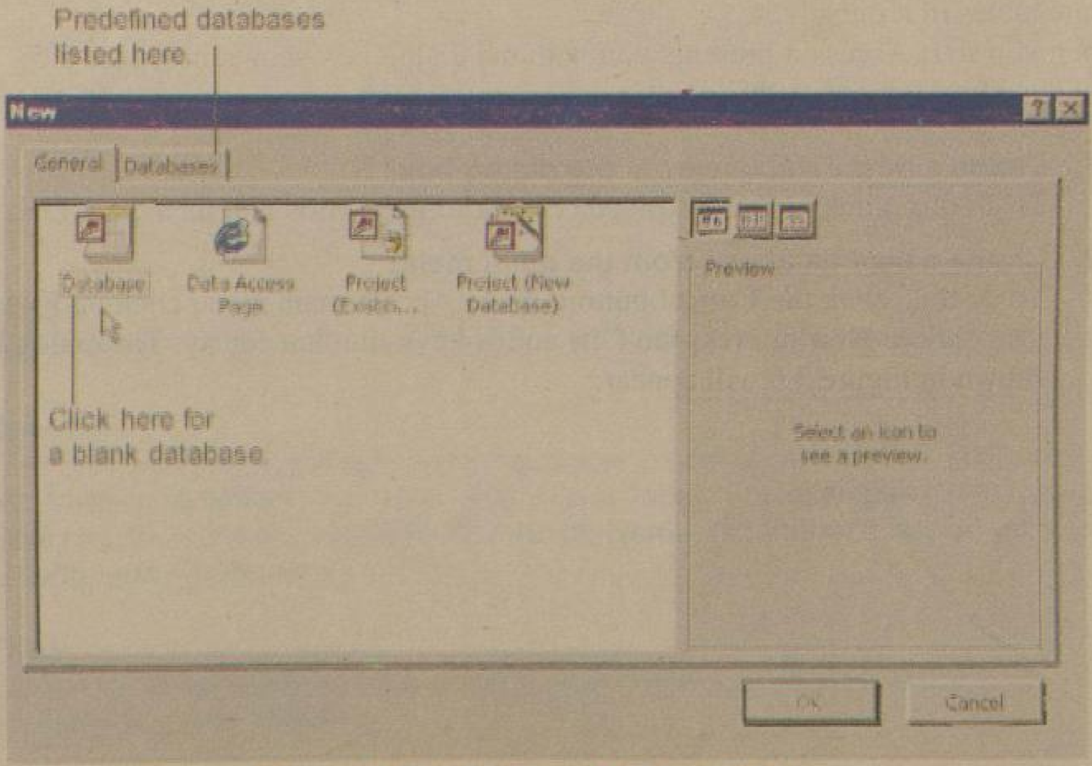

Title bar
Menu bar
Toolbar
Microsoft Access
File Edit View Insert Tools Window Help
Microsoft Access
Create a new database using
○ Blank Access database
○ Access database wizards, pages, and projects
● Open an existing file
More Files...
db1
db2
School
C:\WINDOWS\Desktop\visualmal
OK
Cancel
Ready
NUM

Click here and then OK to create a blank database.

Click here to open a database that already exits.

Click here to go to the main Access window

Figure 3.5: Access Startup window


New
General
Databases
Database
Data Access Page
Project (Existin...
Project (New Database)
Preview
Select an icon to see a preview.
OK
Cancel

Predefined databases listed here.

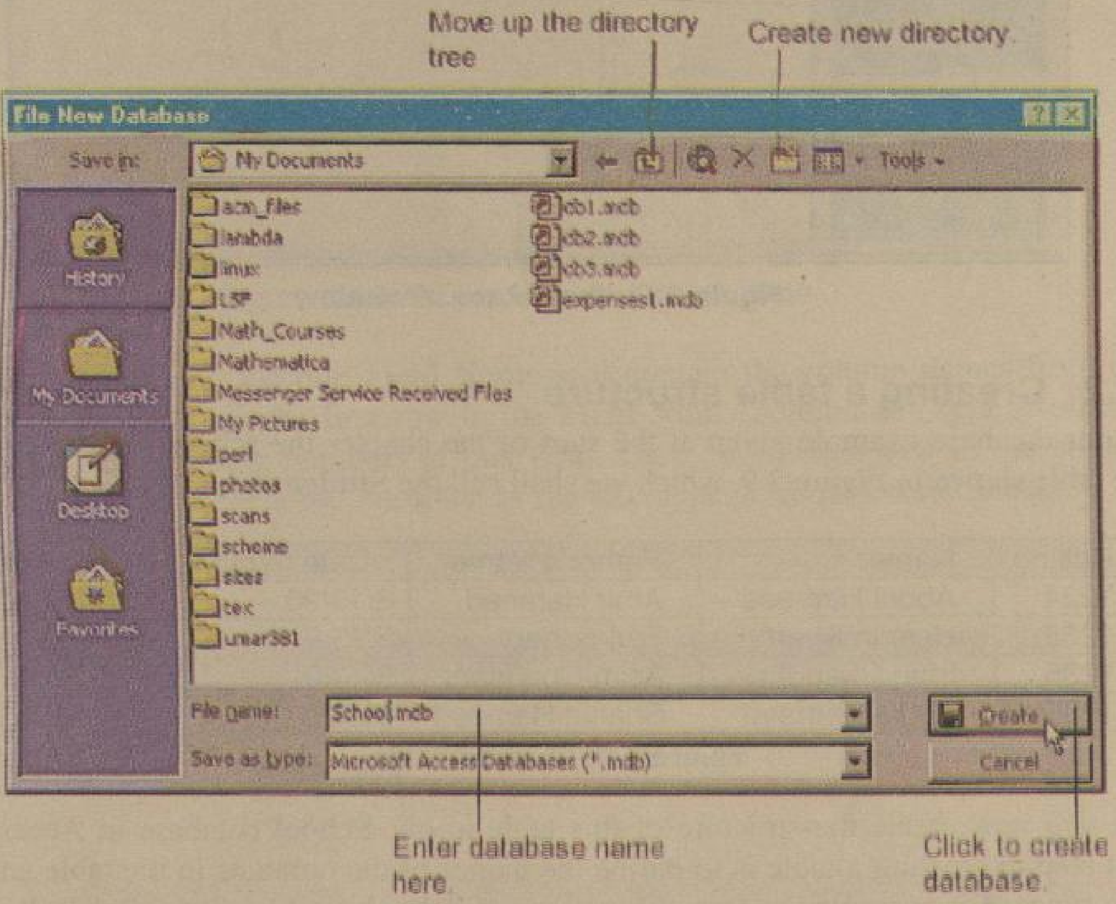
Click here for a blank database.

Figure 3.6: Selecting a blank database

Double click the **Database** icon under the **General** tab.

3. **Open an existing database from the dialog box**
   To do this, choose the **Open an existing file** option and then double click on the any of the existing files presented in the window below.
4. **Open an existing database from the main menu**
   To do this, click the cancel button and then choose **Open** from **File** in the main menu. This will present you with a navigation window, navigate to the directory where the database has been saved and double click on it.

After you have performed one of the above the **File New Database** dialog box will open as shown in Figure 3.7:



Move up the directory tree

Create new directory

File New Database

Save in: My Documents

File name: School.mdb

Save as type: Microsoft Access Databases (*.mdb)

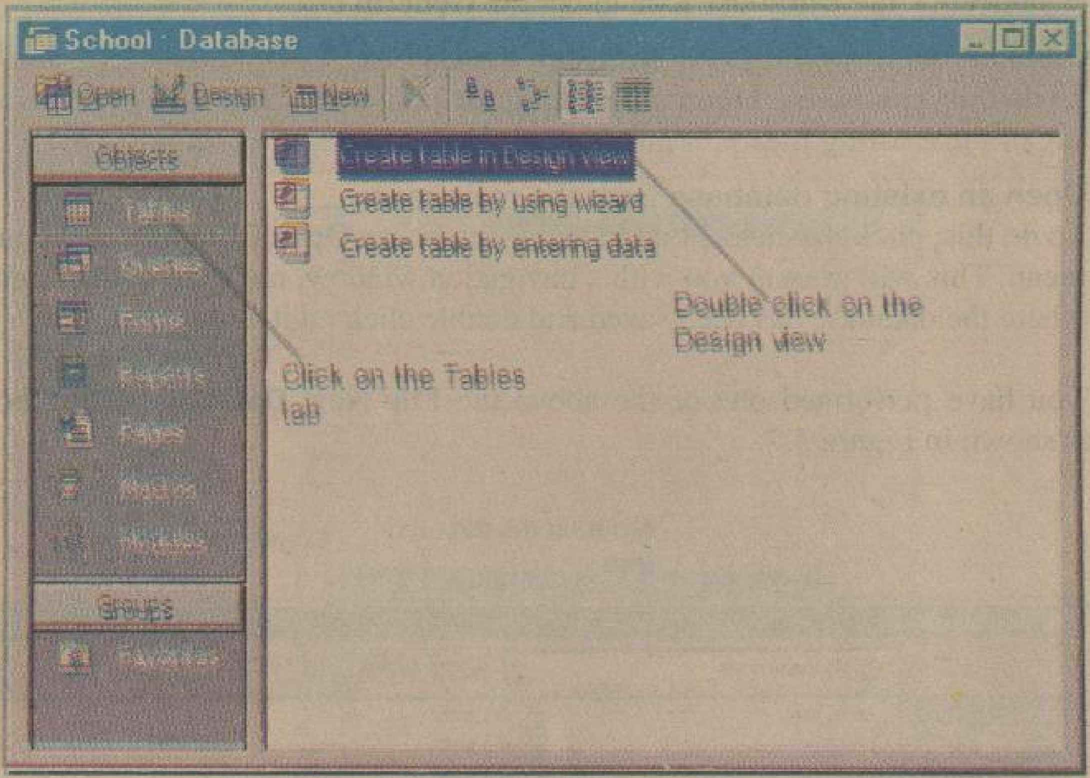
Enter database name here

Click to create database

Tools
History
My Documents
Desktop
Favorites
acm_files
lambda
linux
LSP
Math_Courses
Mathematica
Messenger Service Received Files
My Pictures
perl
photos
scans
scheme
sites
tex
umar381
db1.mdb
db2.mdb
db3.mdb
expenses1.mdb
Create
Cancel

Figure 3.7: Creating a database

Choose the directory where you want to save the new database. Type in an appropriate name for the database e.g. 'School' and click on the **Create** button. This will create the new blank database **School.mdb** (note that Access saves the database with the extension **mdb**). Now you can start adding tables to the database or you can exit Access and come back later to add the tables. If you want to add tables later open this database by clicking on **File** in the main menu, choosing **Open** and selecting the database file from the directory where you saved it. After you have performed the above steps a Database window will appear as shown in Figure 3.8. You are now ready to create new tables.


School : Database
Open
Design
New
Objects
Tables
Queries
Forms
Reports
Pages
Macros
Modules
Groups
Favorites
Create table in Design view
Create table by using wizard
Create table by entering data

Double click on the Design view

Click on the Tables tab

Figure 3.8: The database window

3.2.2 Creating a table structure

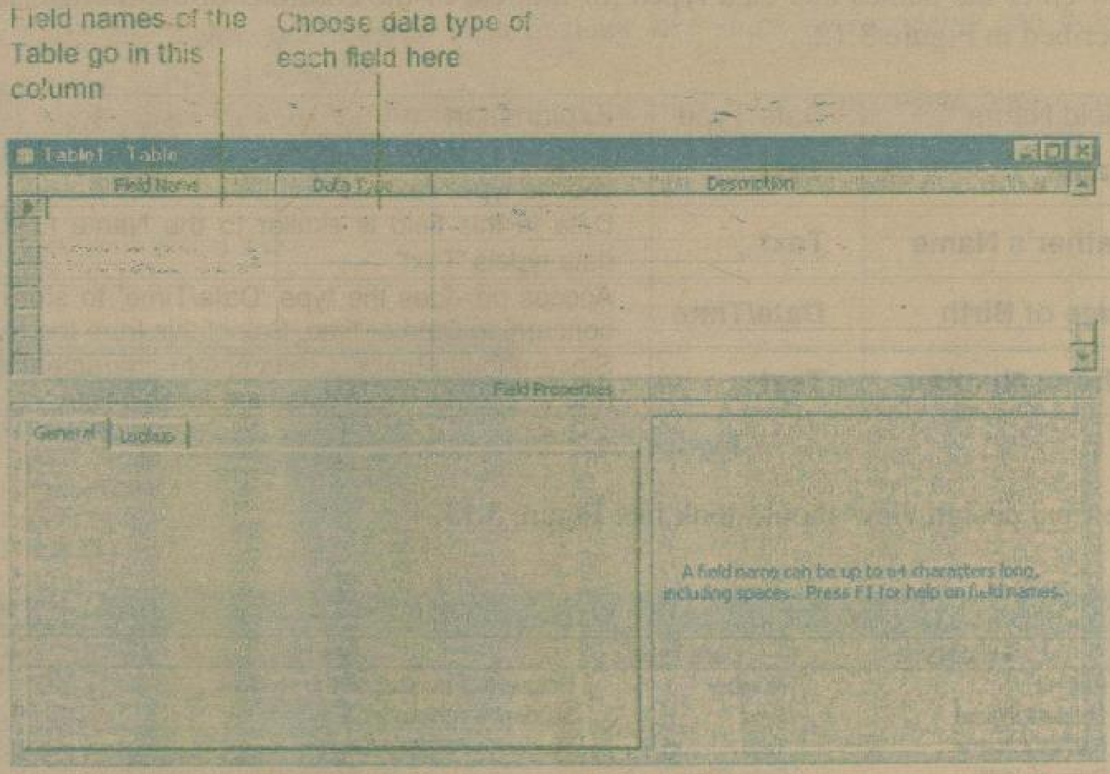
In our database example given at the start of the chapter the school register contained the table shown in Figure 3.9, which we shall call the **Student** table:

Roll no	Name	Father's Name	Date of birth	Phone no
1234	Abdul Hameed	Amir Hameed	3/19/90	586-9830
1235	Usman Khan	Tariq Khan	2/25/91	575-1312
1236	Irfan Chaudhry	Asim Chaudhry	9/4/90	767-1247
1237	Abdul Hameed	Shahid Hameed	4/2/90	555-5590

Figure 3.9: Student Table

We will now create the structure of this table in our **School** database in Access. The first step in creating a table is to define the names of the columns in the table and their data types. For example the first column in our **School** database is named **Roll no** and we intend to store numbers in this field e.g. '1234'. To define the first column in our Access database do the following.

In the Database window, shown in Fig 3.8, double click on **Create table in Design View**. A blank table window like the one shown in Figure 3.10 will appear.



Field names of the Table go in this column

Choose data type of each field here

Table1 : Table
Field Name
Data Type
Description
Field Properties
General
Lookup

A field name can be up to 64 characters long, including spaces. Press F1 for help on field names.

Figure 3.10: Empty table in Design view

The first column with the title **Field Name** will contain the column names from our **Student** table. Click on the first row in the **Field Name** column and type 'Roll no'. Since we will be storing numbers in this field, click on the first row in the column named **Data Type** and choose **Number** as the data type. This process is shown in the Figure 3.11.

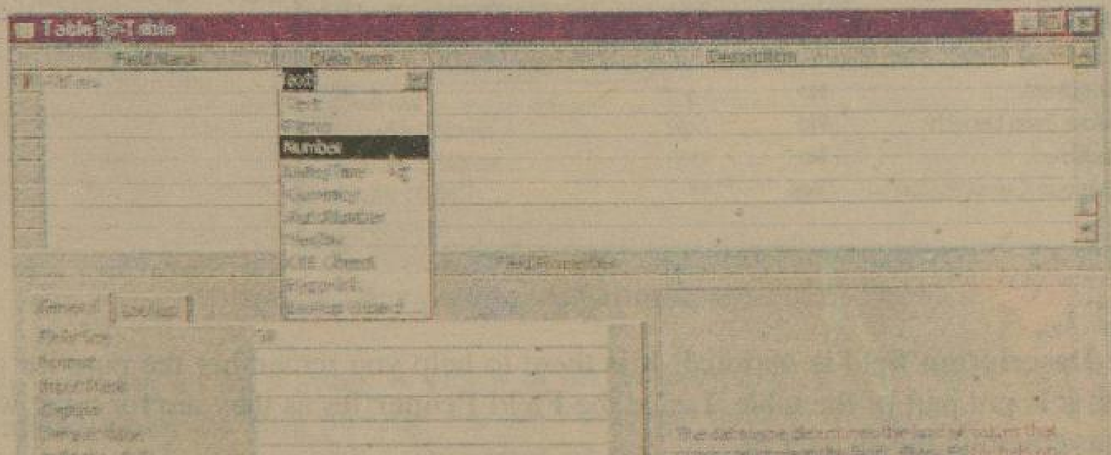


Table1 : Table
Field Name
Data Type
Description
Roll no
Text
Memo
Number
Date/Time
Currency
AutoNumber
Yes/No
OLE Object
Hyperlink
Lookup Wizard
Field Properties
General
Lookup
Field Size
Format
Input Mask
Caption
Default Value
Validation Rule
Validation Text
Required
Allow Zero Length
Indexed
Unicode Compression

The data type determines the kind of values that users can store in the field. Press F1 for help on data types.

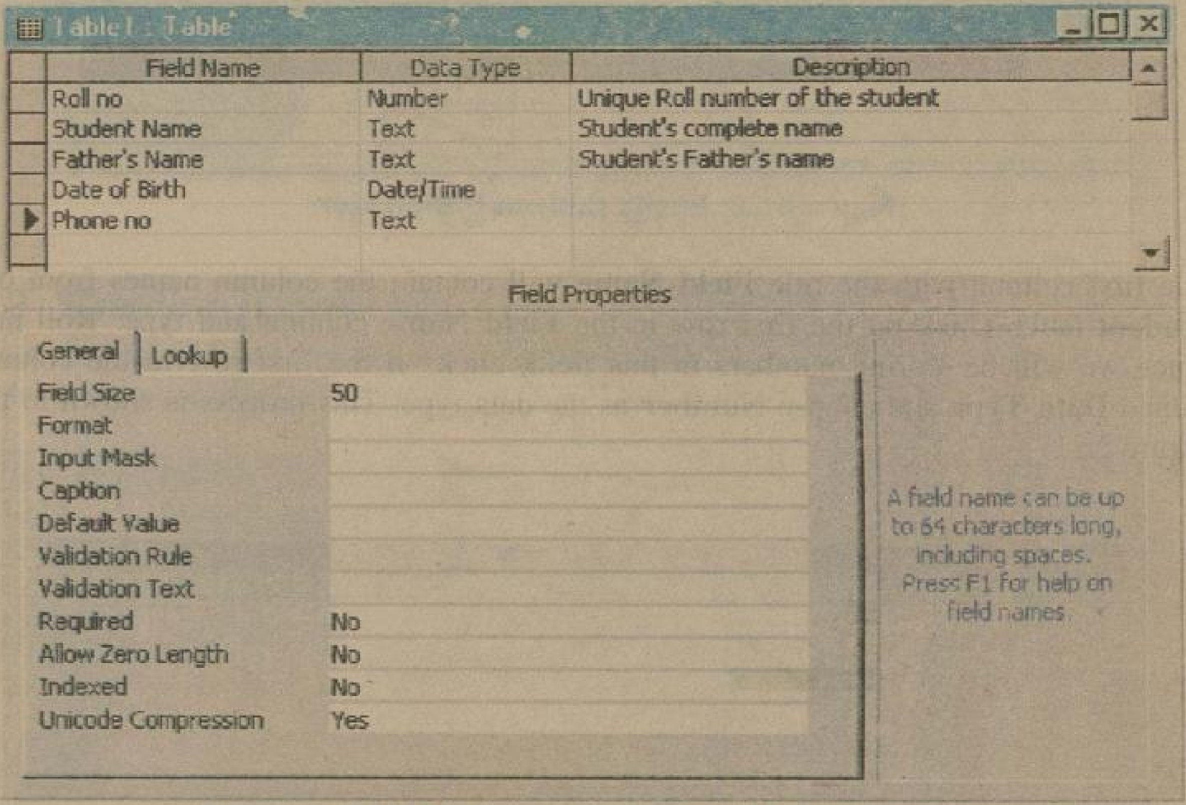
Figure 3.11: Selecting the Data type

Now enter the names and data types for the rest of the columns in the *Student* table as described in Figure 3.12.

Field Name	Data Type	Explanation
Name	**Text**	This field is for the names of students therefore choose type 'Text' .
Father's Name	**Text**	Data in this field is similar to the Name field thus data typeis 'Text'.
Date of Birth	**Date/Time**	Access provides the type 'Date/Time' to store data concerning Date or time. Select this from the menu.
Phone Number	**Text**	Since phone numbers will contain characters like '-' and '#' choose 'Text' data type for this field.

Figure 3.12: Data types

The final design view should look like Figure 3.13.



Field Name	Data Type	Description
Roll no	Number	Unique Roll number of the student
Student Name	Text	Student's complete name
Father's Name	Text	Student's Father's name
Date of Birth	Date/Time	
Phone no	Text	

Field Size	50
Format	
Input Mask	
Caption	
Default Value	
Validation Rule	
Validation Text	
Required	No
Allow Zero Length	No
Indexed	No
Unicode Compression	Yes

A field name can be up to 64 characters long, including spaces. Press F1 for help on field names.

Figure 3.13: Table with field names and data types

The **Description** field is optional. It is there to help you remember the purpose of that field; it is not part of the table. Leave the **Field Properties** as they are for now, we will discuss them in the chapter 5.

3.2.3 Setting the primary key

The last thing left to do is to define the primary key. A ***primary key*** for a table is a field or a group of fields with the property that no two rows of the table contain the same value in that field or group of fields. For example the roll numbers assigned to students in a school are unique; no two rows can contain the same value in the **Roll no** field in the **Student** table, therefore we can set the **Roll no** field as the primary key. On the other hand the **Name** field cannot be a primary key since two students can have the same name e.g. Abdul Hameed.

To set the **Roll no** field as the primary key right click on it in the **Field Name** column and choose **Primary Key** from the menu that appears in Figure 3.14.

Right click on this field and choose Primary key

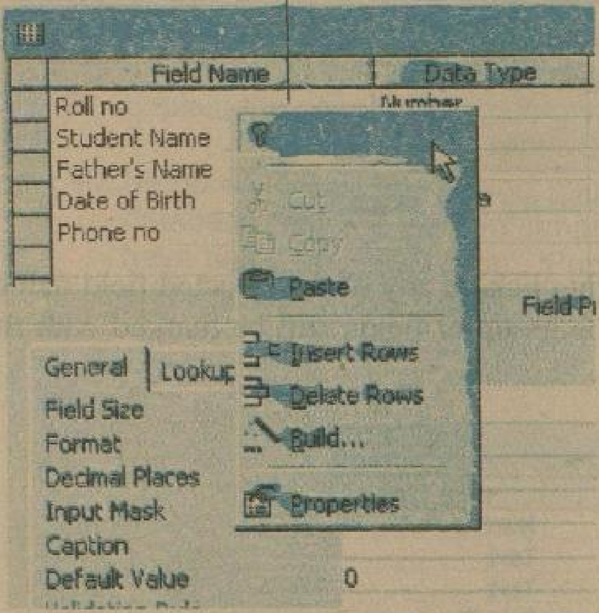
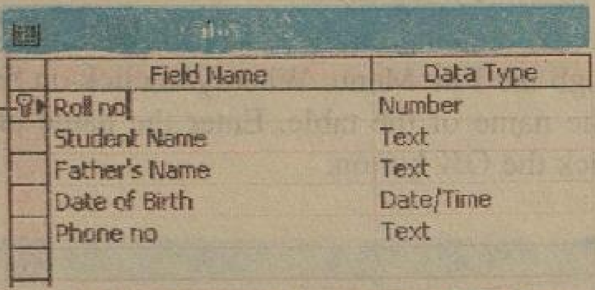

Field Name
Data Type
Roll no
Number
Student Name
Father's Name
Date of Birth
Phone no
Cut
Copy
Paste
Insert Rows
Delete Rows
Build...
Properties
Field Pr
General
Lookup
Field Size
Format
Decimal Places
Input Mask
Caption
Default Value
0

Figure 3.14: Setting the Primary key

The **Roll no** field will be set as the primary key and this will be indicated by the presence of a key symbol next to its name, see figure 3.15.

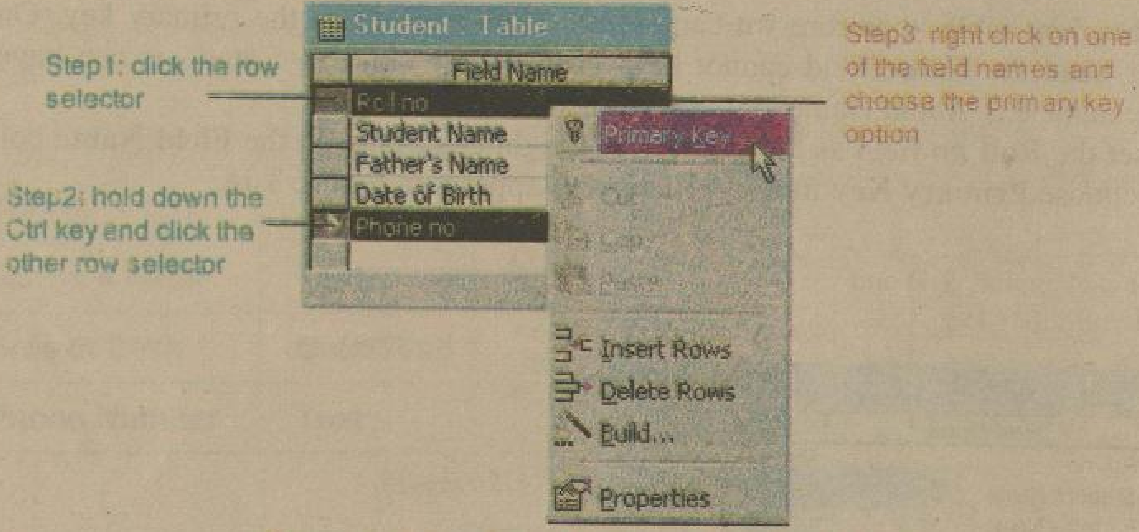
The key symbol indicates that this field is the primary key for the table



Field Name	Data Type
Roll no	Number
Student Name	Text
Father's Name	Text
Date of Birth	Date/Time
Phone no	Text

Figure 3.15: Primary key field identification

You can also set up a group of fields as the primary key. For example we can set the combination of the **Roll no** field and the **Phone no** field as the primary key. To do this click on the ***row selector*** (the grey button on the extreme left of each row) of one of the fields, then hold down the **Ctrl** key and click on the row selectors of the other fields. This process is shown in Figure 3.16.


Student : Table
Field Name
Roll no
Student Name
Father's Name
Date of Birth
Phone no
Primary Key
Cut
Copy
Paste
Insert Rows
Delete Rows
Build...
Properties

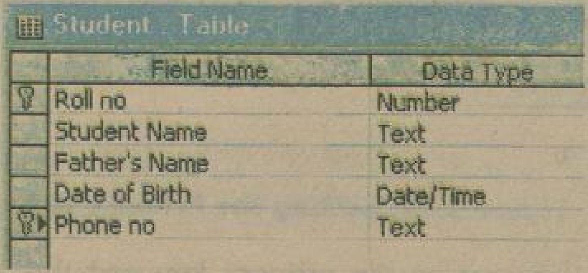
Step 1: click the row selector

Step2: hold down the Ctrl key and click the other row selector

Step3: right click on one of the field names and choose the primary key option

Figure 3.16: Multiple Primary Key Selection

All the fields will be marked as selected. Right click on any of the selected field names and choose **Primary Key** from the menu. The group of fields will be marked with the primary key symbol (Figure 3.17).


Student : Table

	Field Name	Data Type
🔑	Roll no	Number
	Student Name	Text
	Father's Name	Text
	Date of Birth	Date/Time
🔑	Phone no	Text

Figure 3.17

3.2.4 Saving a Table Structure

Save the table structure by clicking on the save icon 💾 in the standard toolbar. You can also save a table through the **File** Menu. When you click on **Save**, a dialog box appears asking you to enter the name of the table. Enter the name of your table in the **Table Name** text box and click the **OK** button.

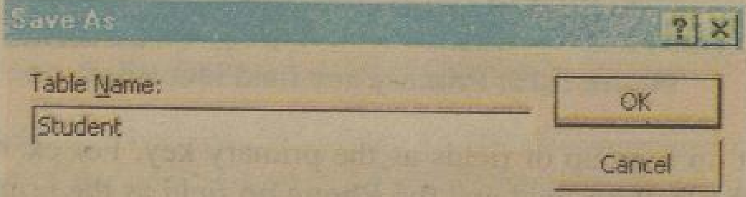

Save As
?
×
OK
Cancel

Table Name: Student

Figure 3.12: Save dialog box

We have now successfully created the structure of a table in the database and saved it. At this moment the table contains no data. Populating the table with data is covered in the next chapter.

Primary key: A unique identifier for the table—that is, a column with the property that, at any given time, no two rows of the table contain the same value in that column.

Exercises

1. Define the following terms:
   Table
   Field
   Record
   Attribute
   Column
   database
2. Outline the procedure for creating a database with tables.
3. Explain the concept of a primary key.
4. Setup a new database and name it 'Library'. Design and create a table in this database that will store information about books in the library. The table should have fields like 'Book Title', 'Publisher', 'isbn no' etc. Can you think of a suitable primary key for this table?

Chapter 4

The Access IDE

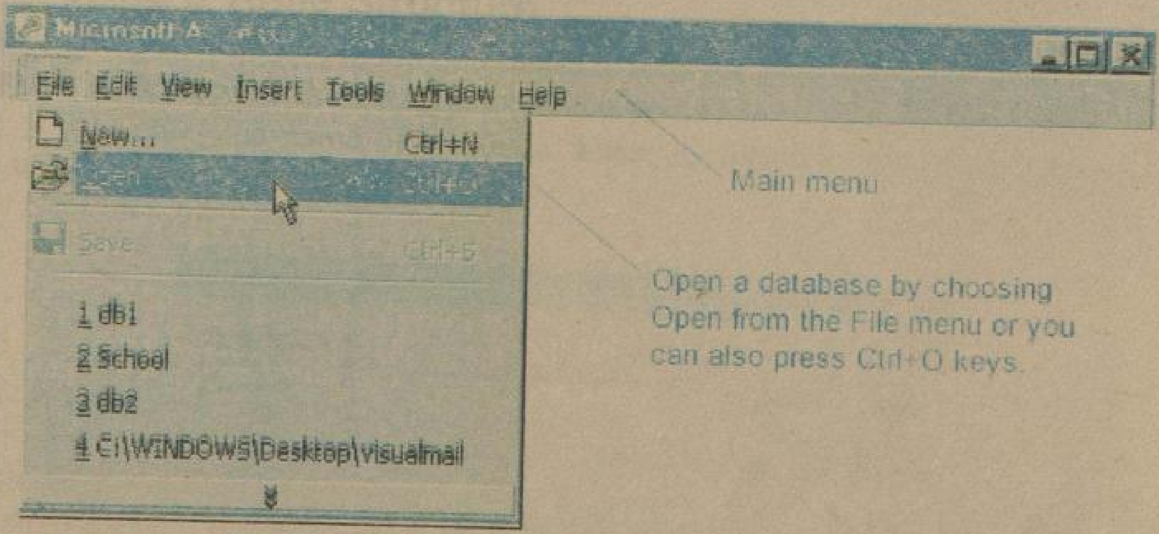
Overview

The major advantage of working in an IDE such as Access is that it provides you with a Graphical User Interface (GUI). The GUI makes tasks, that would otherwise be tedious and time consuming, easy to perform. In this chapter we will look at various features of the Access GUI that will help you in manipulating data in your database tables. This chapter is not going to teach you anything about database principles. Its purpose is to make you comfortable in using and navigating around the Access GUI. At the end of the last chapter we had successfully created an empty table in our **School** database. In this chapter we start with the task of entering data into the table we created.

4.1 Opening a Table

A table that has been created and exists in some database can be opened by performing the steps given below

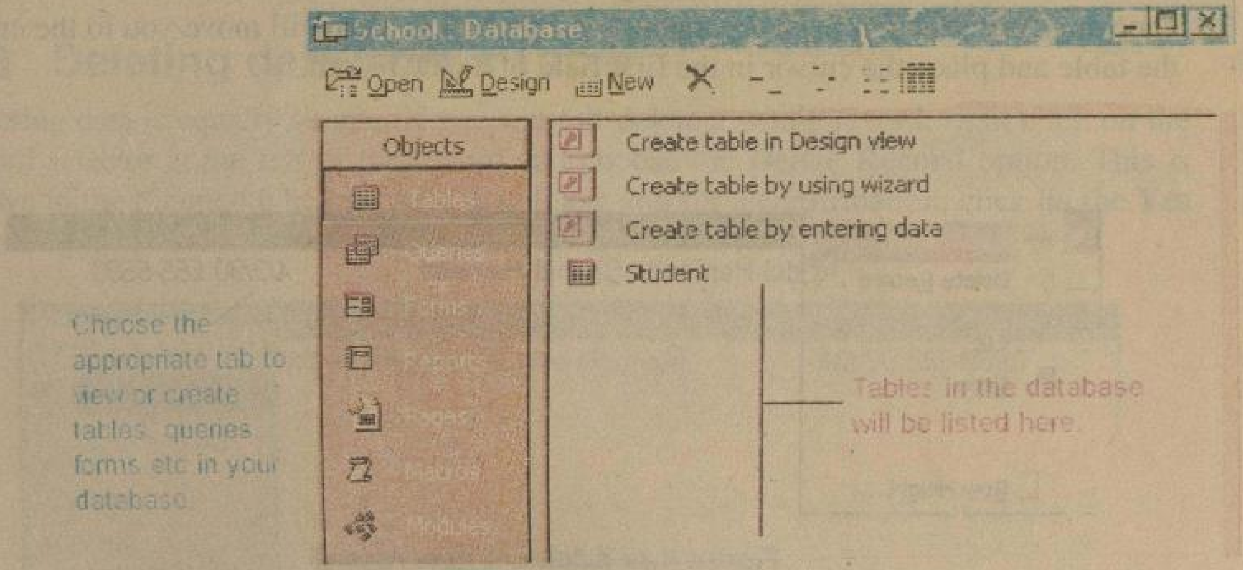
- Open the database that contains the table. You can open an existing database when Access starts by choosing the **Open an existing file** from the startup dialog box as shown in Fig 3.5 of chapter 3. If you are already working in Access choose the **Open** option from **File** in the main menu, see fig 4.1. Navigate to the directory where you saved the database and double click on the file name (it will have the extension **mdb**).



Open a database by choosing Open from the File menu or you can also press Ctrl+O keys.

Figure 4.1: Opening a table from main menu

- After you have opened the desired database, Access will present you with the **Database** window. In the database window click on the **Tables** tab. The names of the tables in your database will appear in the window as shown in


School : Database
Open
Design
New
Objects
Tables
Queries
Forms
Reports
Pages
Macros
Modules
Create table in Design view
Create table by using wizard
Create table by entering data
Student

Choose the appropriate tab to view or create tables, queries forms etc in your database

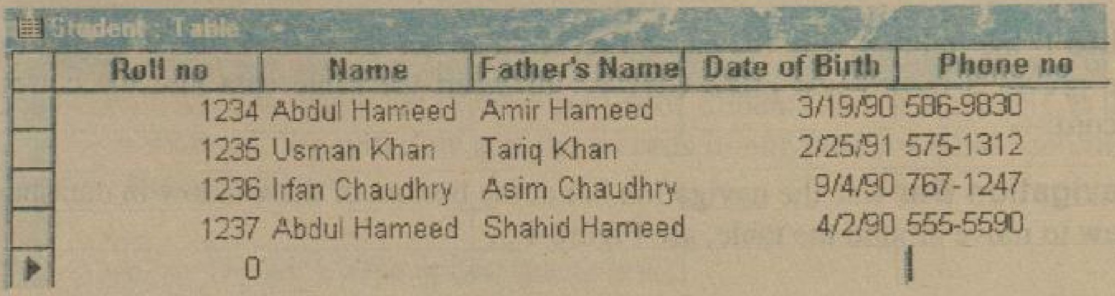
Tables in the database will be listed here

Figure 4.2: Opening table from the database window

- To open the table for entering or editing data, double-click the table name. This will take you into the **Datasheet view**.
- To view or change the table structure, click on the table name and then click on the Design button Design. This will take you into the **Design view**.

4.2 Adding Data to a Table

Open the table in Datasheet view as shown in the last section. We will now enter the data that is shown in the fig 4.3 below and then save the table.


Student : Table

	Roll no	Name	Father's Name	Date of Birth	Phone no
	1234	Abdul Hameed	Amir Hameed	3/19/90	586-9830
	1235	Usman Khan	Tariq Khan	2/25/91	575-1312
	1236	Irfan Chaudhry	Asim Chaudhry	9/4/90	767-1247
	1237	Abdul Hameed	Shahid Hameed	4/2/90	555-5590
▶	0				

Figure 4.3 : Sample data for the Student table

To enter data into the table click on the first field of the first record, type in the data and press the **Enter** button. Access will automatically move the cursor to the next field of the current record. When all the fields of the first record are filled, Access will move the cursor to the first field of the next record.

Note that to the left of each record there is a button ☐, this is the **record selector**. If you click on it the entire record will be selected also the current record has a record selector button with an arrow head on it ▶.

If you open a table that already has a large number of records in it, you can add a new record from anywhere in the table by right clicking on any record selector button and choosing **New Record** from the menu, see Figure 4.4. This will move you to the end of the table and place the cursor in the first field of a new record.

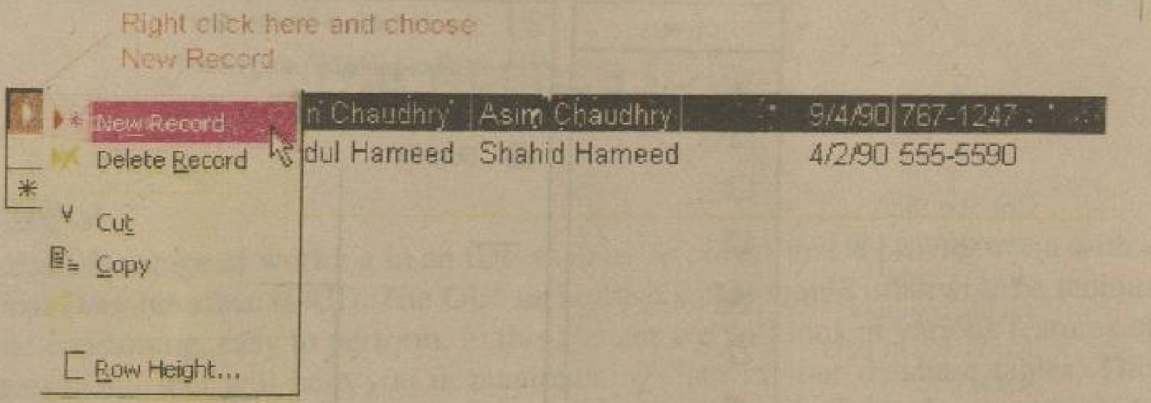

Right click here and choose
New Record
New Record
Delete Record
Cut
Copy
Row Height...
n Chaudhry Asim Chaudhry 9/4/90 767-1247
dul Hameed Shahid Hameed 4/2/90 555-5590

Figure 4.4: Adding a new record

4.2.1 Tips for Adding Data

Here are some tips that will make it easier for you to add data to the table.

- **Tab** Use the tab button to move between fields. If you want to leave a field empty just press the tab or enter key.
- **AutoNumber** Field with the data type set to **AutoNumber** is automatically filled by Access each time a new record is entered. Do not change or put values in fields of type autonumber.
- **Date/Time** Access has shortcut keys to enter the current time or the current date. To enter the current time press **Ctrl + :** (that is the control key and the character ':') and to insert the current date press **Ctrl + ;** .
- **Ditto** If the field entry in the current record matches the entry in the same field of the previous record just press **Ctrl + '** to insert the same data into the current record.
- **Navigation Bar** Use the navigation bar at the bottom of the window in datasheet view to move around the table, see Figure 4.5.

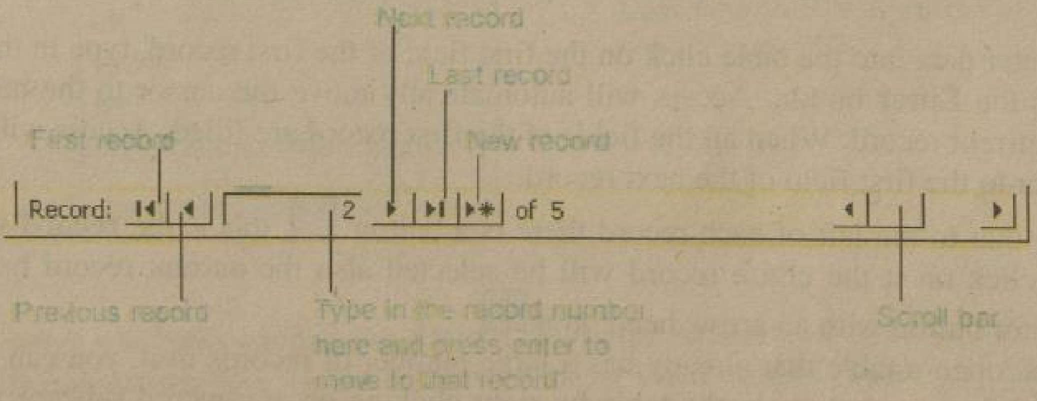
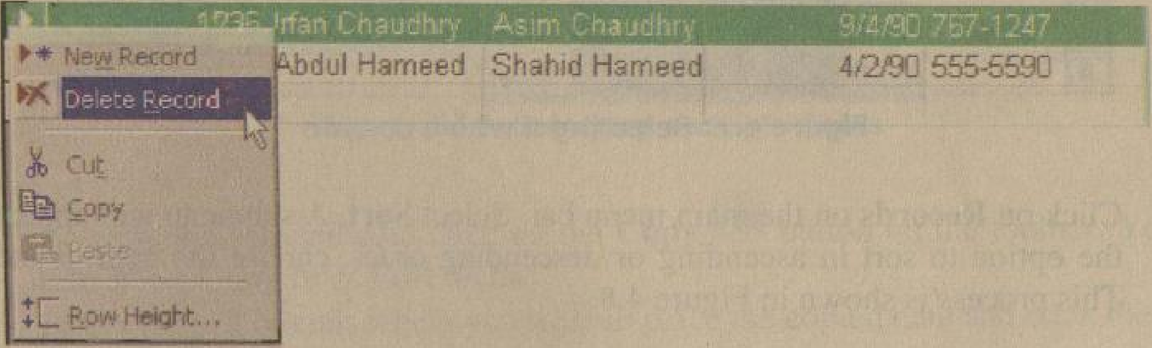

Next record
Last record
New record
First record
Record: 2 of 5
Previous record
Type in the record number here and press enter to move to that record
Scroll bar

Figure 4.5: Using the navigation bar

4.3 Deleting data from a Table

Deleting data is equally simple. If you want to delete a whole record, right click on the record selector at the left of the record and choose the **Delete Record** option. This is shown in the Figure 4.6 below. Access will ask you for a confirmation, click on the **Yes** button to remove the record.



1236	Irfan Chaudhry	Asim Chaudhry	9/4/80 757-1247
	Abdul Hameed	Shahid Hameed	4/2/90 555-5590

Figure 4.6: Deleting a Record

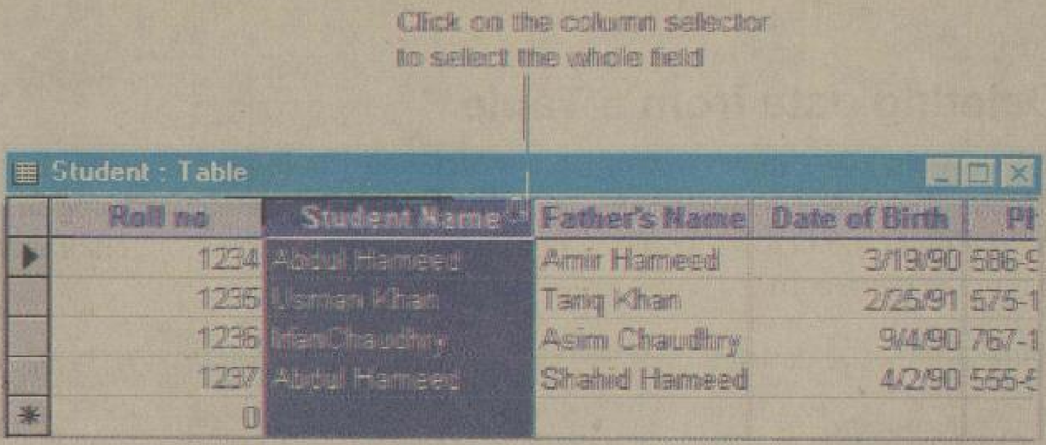
To delete / replace data in a field, place the mouse pointer near the left end of a field, the pointer will change to a large plus sign ✚, click on the field to choose the entire field. Type in the new data you want in the field or press the delete button to erase the data in the field.

If you want to restore the deleted data, press the **Esc** button on the keyboard or click the undo ↶ toolbar button.

4.4 Sorting Data

It is often desirable to view the data within your table in some meaningful order e.g. alphabetical or numeric. When data is needed in a specific order, we sort the records in a table based on the data in a field e.g. in a table which has the test scores of students stored in a field, you may want to view the list of students sorted according to the marks they have obtained. We will now sort the records in our **Student** table according to the **Student Name** field.

- Open the **Student** table in datasheet view.
- Move the cursor to the field name at the top of the column. The cursor will change to an arrow pointing downwards ⬇. Click once, the whole column will be selected, see fig 4.7.



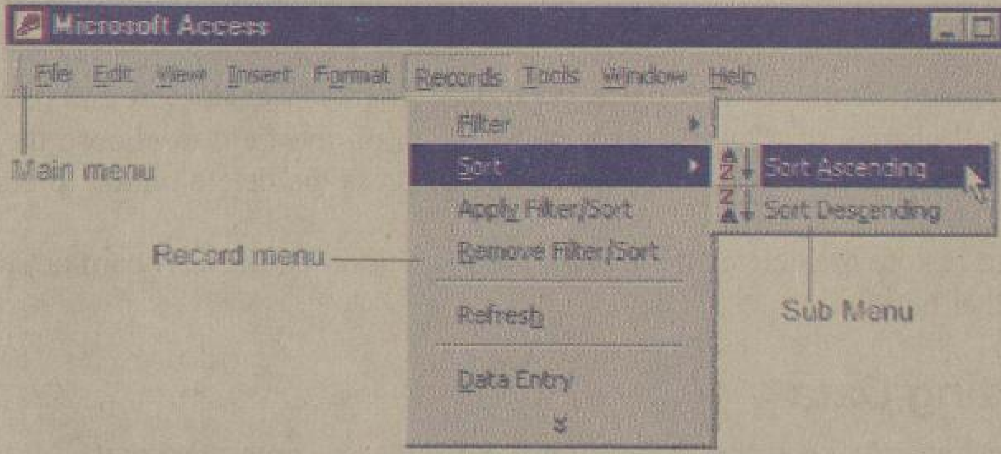
Click on the column selector to select the whole field

Student : Table

Roll no	Student Name	Father's Name	Date of Birth	Ph
1234	Abdul Hameed	Amir Hameed	3/19/90	586-9
1235	Usman Khan	Tariq Khan	2/25/91	575-1
1236	IrfanChaudhry	Asim Chaudhry	9/4/90	767-1
1237	Abdul Hameed	Shahid Hameed	4/2/90	555-5
0				

Figure 4.7: Selecting a whole column

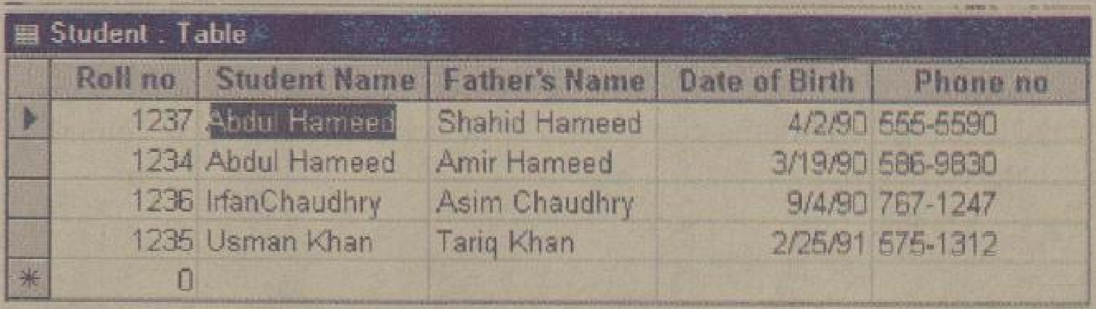
- Click on **Records** on the main menu bar. Select **Sort**. A submenu will appear with the option to sort in ascending or descending order, choose the appropriate one. This process is shown in Figure 4.8.



Microsoft Access
File Edit View Insert Format Records Tools Window Help
Filter
Sort
Apply Filter/Sort
Remove Filter/Sort
Refresh
Data Entry
Sort Ascending
Sort Descending
Main menu
Record menu
Sub Menu

Figure 4.8: Choosing the sort order

- The Field selected will be sorted in the chosen order. Fig 4.9 shows the table sorted according to the **Student Name** field. Note that the roll numbers are no longer in ascending order.



Student : Table

Roll no	Student Name	Father's Name	Date of Birth	Phone no
1237	Abdul Hameed	Shahid Hameed	4/2/90	555-5590
1234	Abdul Hameed	Amir Hameed	3/19/90	586-9830
1236	IrfanChaudhry	Asim Chaudhry	9/4/90	767-1247
1235	Usman Khan	Tariq Khan	2/25/91	575-1312
0				

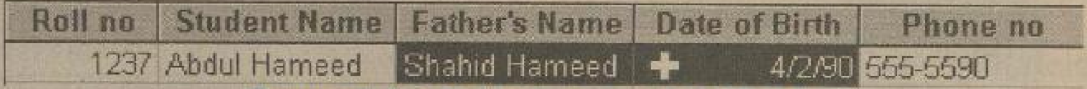
Figure 4.9: Records sorted according to the 'Student Name' field

- You can also sort your data with the help of toolbar. The icon AZ↓ sorts the data in ascending order, while ZA↓ sorts the data in descending order.

4.5 Copying and Moving Data

To copy data:

- Select the cells that you want to copy. You can select multiple cells by placing the mouse pointer near the left end of the leftmost field so that it changes to a plus sign, drag it to select the fields you want copied. See figure 4.10



Roll no	Student Name	Father's Name	Date of Birth	Phone no
1237	Abdul Hameed	Shahid Hameed	4/2/80	555-5590

Figure 4.10: Selecting multiple fields

- To copy the selected data click on the **Copy** button on the toolbar. You can also select **Copy** from **Edit** menu.
- Select the cell or cells where you want to place the copied data and click the **Paste** button on the toolbar or use the **Paste** option in the **Edit** menu. Note that the number of cells that you selected for copying should equal the number of cells you want to paste the data in, otherwise some data maybe lost.

To Move Data:

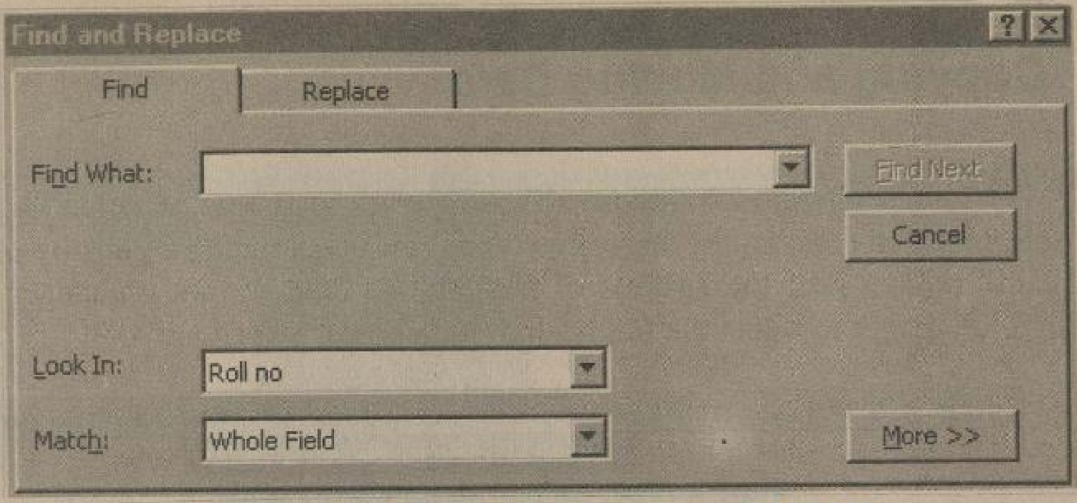
- To move data use the same procedure as described in the last section but choose the cut button from the toolbar instead of the copy button.

4.6 Locating and Replacing Data

When a small amount of information is to be stored, say 10 records, the advantages of a database management system are not apparent. We can store the information in a text file and manipulate it manually. The need for a database becomes apparent when the amount of information becomes large. In a real life implementation there maybe millions of records in a database table. In a real life school database, there may be a 1000 or more records in the student table. Locating or replacing some specific information e.g. a specific phone no, in a database of even this size becomes very difficult. The Access GUI provides useful functions to locate and replace information in a table. We study these functions in this section.

4.6.1 Finding Data

To find data in a specific field, click on the field and click on **Find** from the **Edit** menu. The **Find and Replace** dialog box will open as shown in Figure 4.11.



Find What:

Look In: Roll no

Match: Whole Field

Figure 4.11: The Find and Replace dialog box

Enter the data you want to search for in the **Find What** text box. The field on which you clicked will be displayed by default in **Look In** list box. If you want to change the search range from one field to entire table, click on the arrow on the right of the **Look In** list box and choose the name of the table, see Figure 4.12.

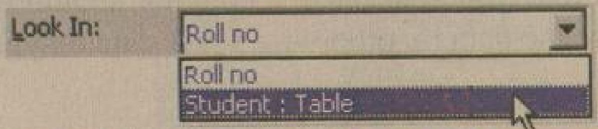


Figure 4.12: Selecting the whole table for search

The Match list box gives you options for the way the in which the given string is to be matched. You have the three choices shown in Figure 4.13.

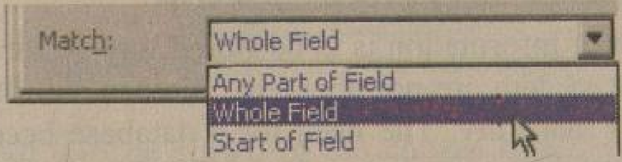


Figure 4.13: Selection of Match criteria

1. **Whole Field:** This locates only those entries that exactly match your search string.
2. **Any Part of Field:** This locates strings that have the search string you gave as a sub-string. If you typed in '12' in the **Find What** text box it will match any string containing '12' as a sub-string e.g. 1234, 121, 3421 but not 12 or 132.
3. **Start of Field:** This locates strings that begin with the search string you typed in. The search string '12' will match 1234, 124, 12 but not 1123, 1112.

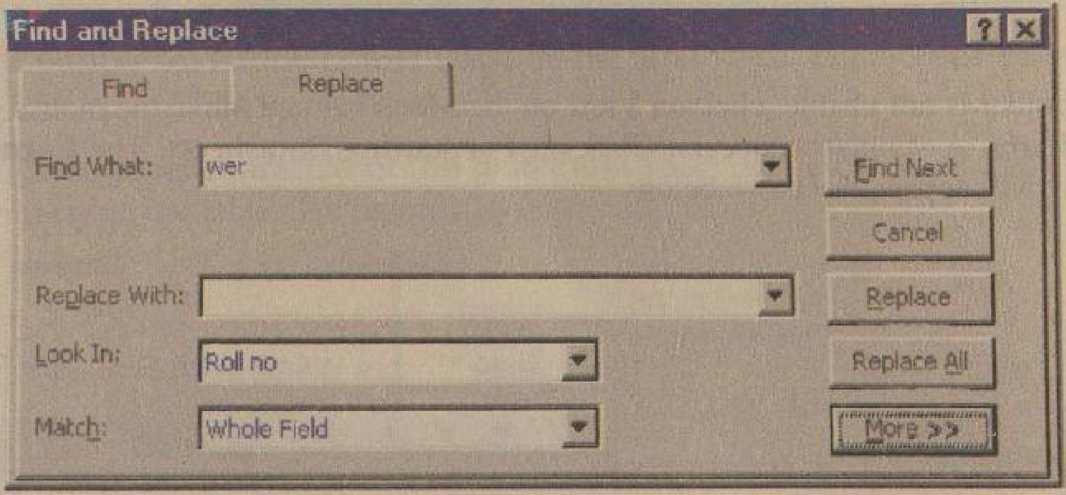
Click on the **More** button to view additional options. The **Search** list box gives you three options. Choose **All** if you want to search the entire table. Choose **Up** if you want to search only the records upwards from the currently selected record. Choose **Down** if you want to search downwards from the currently selected record.

To find the records that match your search string, click on the **Find Next** button. Each time you click on the **Find Next** button, Access will move to the next record field that matches the search string.

4.6.2 Replacing Data

Replacing data consists of two steps, finding the data that matches the search string and replacing the matched data with the given replacement string. To replace data within your table perform the following steps:

- If you want to replace data in a certain field of all the records click on that field. If you want to replace data throughout the table click on any field.
- Click on **Edit** on the main menu and choose **Replace**. The dialog box shown in Figure 4.14 will appear.



Find What: wer

Replace With:

Look In: Roll no

Match: Whole Field

Figure 4.14: Replace dialog box

- All the fields in this dialog box are the same as the one in the last section on finding data except the **Replace With** field. Enter the text in the **Replace With** field that should replace the data that matches the search string e.g. if you want to change the spellings of the name ‘Omer’ to ‘Omar’, type in ‘Omer’ in the **Find What** text box and ‘Omar’ in the **Replace With** text box.
- Select rest of the fields as shown in the last section.
- Click on **Find Next** to go to the first field that matches the search criteria. If you want to replace the data in this particular field click on **Replace**, Access will replace the data and move to the next field that matches the search criteria. You can also replace all the matching data in one step by clicking on **Replace All**.

4.7 Formatting a Table

Access provides easy to use functions to make the appearance of the data displayed in a table in the datasheet view more attractive. The next two sections explain how.

4.7.1 Formatting Data

To Format your data, make sure that the Formatting toolbar shown in Figure 4.15 is visible.

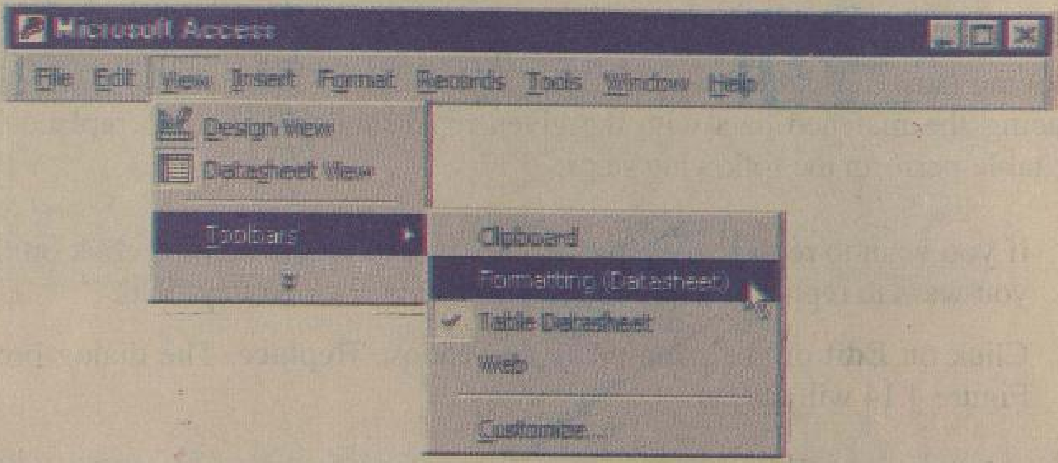

Microsoft Access
File
Edit
View
Insert
Format
Records
Tools
Window
Help
Design View
Datasheet View
Toolbars
Clipboard
Formatting (Datasheet)
Table Datasheet
Web
Customize...

Figure 4.15: Viewing the Formatting toolbar

If the toolbar is not visible, click on **View** on the Menu bar and move the mouse pointer to **Toolbars.** A submenu appears; Click on **Formatting(Datasheet)**. The **Formatting Toolbar** will appear, as shown below.

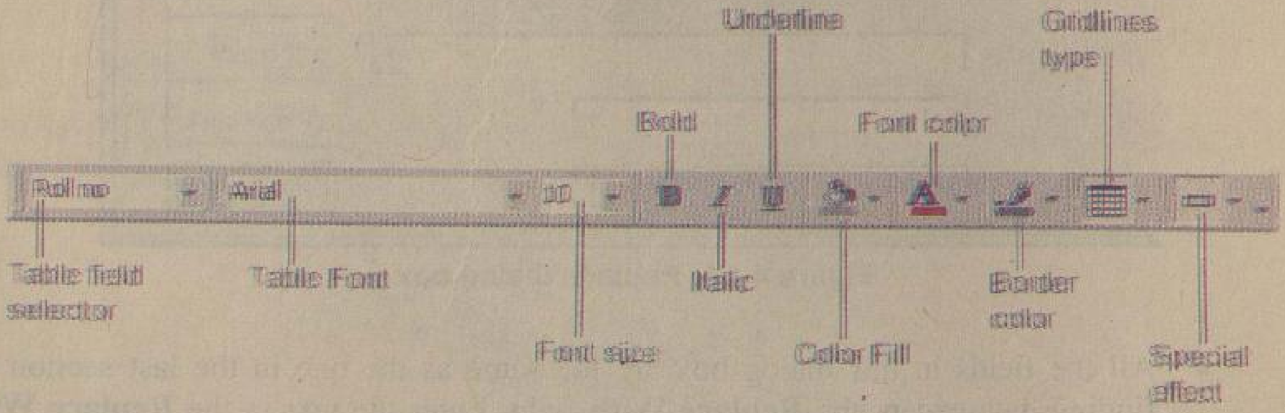

Underline
Gridlines type
Bold
Font color
Rollno
Arial
10
Table field selector
Table Font
Italic
Border color
Font size
Color Fill
Special effect

Figure 4.16: Formatting Toolbar

Once the Formatting toolbar is visible you can easily do the following:

- **Change the table text font:** click on the down arrow of the Font text box. A large number of options appear. Choose the font style. The font of the whole table will be changed.
- **Select font size:** choose the font size from the drop down font size list.
- **Boldface, Italicize or Underline text:** click on the buttons labeled **Bold**, **Italic** and **Underline** in the Figure 4.16 above. Note these options will affect the whole table. You can undo any of these effects by clicking on the corresponding button again.

- **Change table background color:** You can use the **Color Fill** drop down box to choose the background color for your table. Click on the arrow next to the button to see all the available colors.

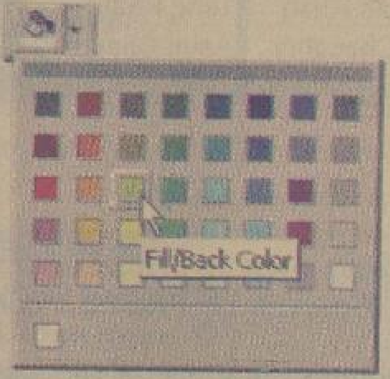

Fill/Back Color

Figure 4.17: Background color selection

- **Change text font color:** Click on the arrow next to the **Font color** button. Choose a color by clicking on it.
- **Change appearance of the gridlines:** There are four ways of displaying grid lines in your table. See figure 4.18 for details.

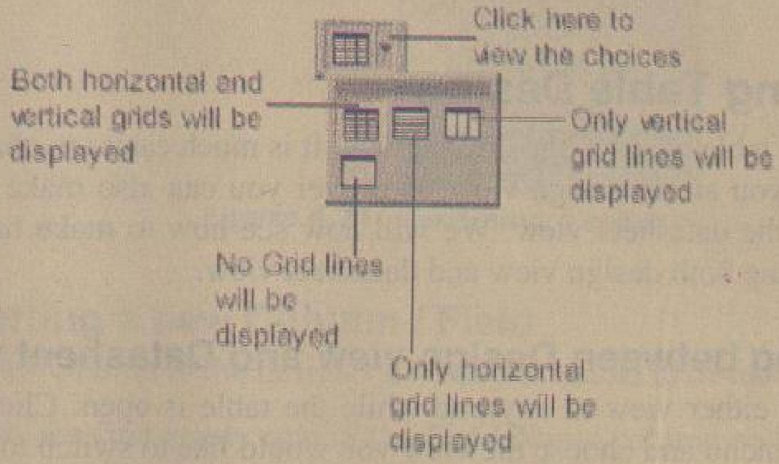
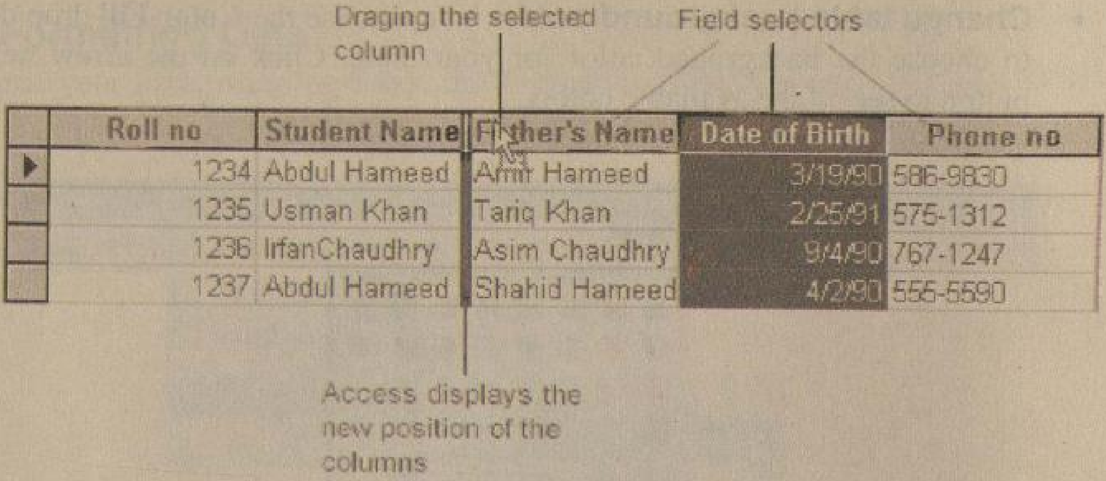

Click here to view the choices
Both horizontal and vertical grids will be displayed
Only vertical grid lines will be displayed
No Grid lines will be displayed
Only horizontal grid lines will be displayed

Figure 4.18: Gridline format choices.

4.7.2 Arranging Table Columns

While in datasheet view you may wish to view the table fields in a certain order. For example, it might be desirable in our **Student** table to have the **Date of Birth** field next to the **Student Name** field. Let us see how this can be achieved.

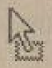
- Select the column that you want to move by clicking the field selector at the top of the column. If you want to select multiple columns drag the mouse pointer through several field selectors. To deselect selected columns, click the data area of any column.


Draging the selected column
Field selectors
Access displays the new position of the columns

	Roll no	Student Name	Father's Name	Date of Birth	Phone no
▶	1234	Abdul Hameed	Amir Hameed	3/19/90	586-9830
	1235	Usman Khan	Tariq Khan	2/25/91	575-1312
	1236	IrfanChaudhry	Asim Chaudhry	9/4/90	767-1247
	1237	Abdul Hameed	Shahid Hameed	4/2/90	556-5590

Figure 4.19: Moving a column

- To move the selected columns click one of the selected field selectors and drag the selection to the left or right. As you move the pointer a bold black line will mark the place where the selected columns will be moved. Release the mouse button to place the selected columns in the desired spot. This process is shown in Figure 4.19. Note that the mouse pointer will change to the form shown below while you are dragging the selection.



4.8 Changing Table Design

When we created a table we used the design view. It is much easier to change the design of the table when you are in design view. However you can also make changes to the table design from the datasheet view. We will now see how to make basic changes to the table design using both design view and datasheet view.

4.8.1 Switching between Design view and Datasheet view

You can switch to either view at anytime while the table is open. Click on the **View** button in the main menu and choose the view you would like to switch to. This is shown in the Figure 4.20 below.

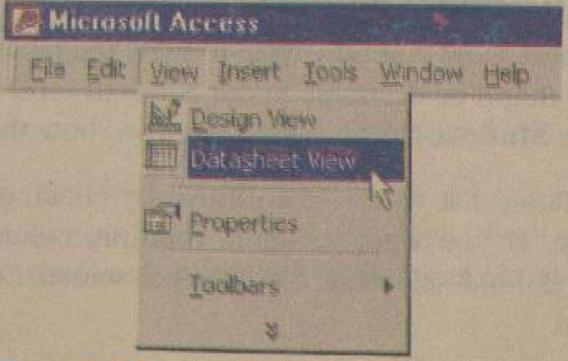

Microsoft Access
File Edit View Insert Tools Window Help
Design View
Datasheet View
Properties
Toolbars

Figure 4.20: Switching between views

4.8.2 Changing a field name

- In design view click on the table field name you want to change in the **Field Name** column of the design view and edit the text.
- In datasheet view double click on the field selector at the top of the column. The field name will be selected and an **I-beam** pointer appears. Change the name according to your requirements.
- You can also change the name by right-clicking on the field name you want to change. The menu shown in fig 4.21 will appear, select **Rename column** . The column will be selected and you will be able to change the field name.

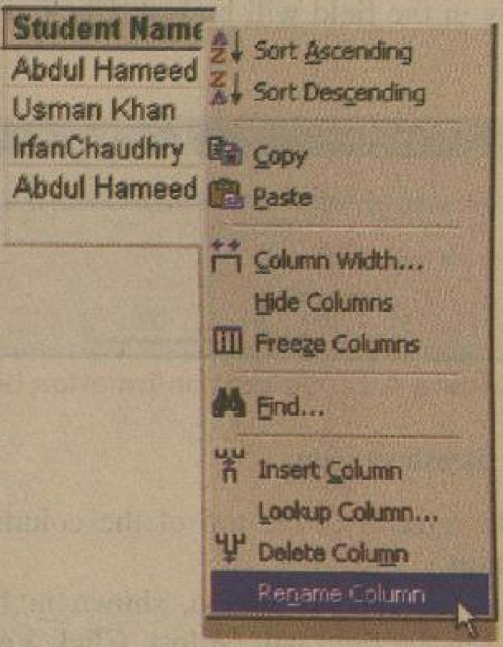

Student Name
Abdul Hameed
Usman Khan
IrfanChaudhry
Abdul Hameed
Sort Ascending
Sort Descending
Copy
Paste
Column Width...
Hide Columns
Freeze Columns
Find...
Insert Column
Lookup Column...
Delete Column
Rename Column

Figure 4.21: Renaming a column

4.8.3 Inserting a new Column / Field

To insert a new column in the design view open the table in design view and then

1. Click on the first empty row in the **Field Name** column and type the name of the new column. If you want to insert the new field between two existing fields, right click on the top field and choose **Insert rows** from the menu. A blank row will appear between the two existing fields, fill in the name of the new column in the **Field Name** column.
2. Choose the appropriate data type in the **Data Type** column.
3. Type in optional comments about the new column in the **Description** field.

To insert a new column in the datasheet view switch to the datasheet view and then:

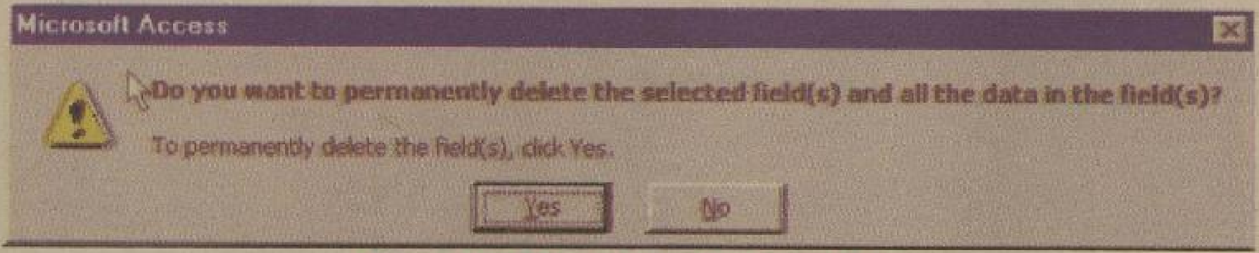
1. Right click on the field selector at the top of a column and select **Insert Column.**
2. A new column will be inserted to the left of the column you selected. Rename the column. The new column will have the data type **Text** as default. If you want to change this you will have to switch to the design view and do so.

4.8.4 Deleting a Column

Deleting a column means deleting the column from the table structure and deleting the data within the column as well, therefore you should be very careful while deleting a column. What's more, you can't undo the column deletion.
To delete a column in the design view

1. Right click on the row selector ▶ of the column that you want to delete. Select the delete rows option.

2. Access will ask you for a confirmation, shown in Figure 4.22 below, with a warning that all data in the field will be lost. Click yes if you want to delete the column.



Do you want to permanently delete the selected field(s) and all the data in the field(s)?

To permanently delete the field(s), click Yes.

Figure 4.22: Delete confirmation box

To delete a column in the datasheet view

1. Right click the field selector at the top of the column you want to delete and choose **Delete Column**.
2. Access will ask you for a confirmation, shown in Figure 4.22 above, with a warning that all data in the field will be lost. Click **Yes** if you want to delete the column.

Exercises

1. What extensions are databases files given by Access. (Extension is the part of the file name that occurs to the right of the '.' For example executable programs are given the extension 'exe' in windows)

2. What are the two *table views* available in access? How do you switch between them?

3. Where is the *record selector* button located and what is its purpose?

4. How do fields with data type AutoNumber differ from fields of other types?

5. When we showed you how to sort records in a table, the sorting was done based on a data in a single field of the table. Instead sorting on a single field try to sort

records in a table based on two fields. For example sort the records in ascending order in the *Student* table by selecting both the *Student Name* and *Father's Name* fields. Can you see how the records are being sorted? (hint: remember we have two records with the *Student Name* field containing 'Abdul Hameed', which of these records comes first in the sorting order and why?)

Chapter 5

Database Design

Overview

One of the major reasons we store data in database systems is our need for efficient space utilization and faster data manipulation. Moreover, the database system automatically verifies that the data being input meets certain specifications. Hence data is stored in a consistent manner. Database design plays a critical role in achieving these goals of efficiency, speed and consistency. Poor database design will result in poor performance, no matter what database management system you use. This chapter covers some basic database design principles, which will help you to tune your database for optimal performance.

5.1 Concept of a Relational Database

You should by now have an understanding of the concept of a table. It is useful to think of a table as a ***data structure***. An important fact about relational databases, such as Microsoft Access and Oracle, is that they may store the data on the hard disk in different formats but all of them represent the data to the user in the form of a table data structure. Relational databases also provide various *operations* that the user can use to manipulate the data in the database tables. For example, the database access language provides a way to select data from one or more tables that fits a certain criteria.

We can use the data structures and the operations that various database systems provide to the user to categorize these systems. Employing this criteria, database systems can be divided into four broad categories, namely **relational**, **hierarchic, inverted list**, and **network** systems. As we have seen, relational systems present data to user in form of tables and provide various operations that operate on these tables. Similarly, a hierarchic system presents the data to the user in form of a set of tree structure, and also provides operations for manipulating such structures.

In the past 20 years almost all the commercial database systems developed can be categorized as relational systems. Most of the research during this time in the database field has also been on the relational database model. However, more recently the research has focused on a variety of systems e.g. **Deductive**, **Expert**, **Object-oriented** and **Semantic**. Some of these have been derived from the relational model while others are a step in an entirely new direction. We will only concentrate on the relational model throughout this book.

We are now ready to define the concept of a ***Relational Database***.

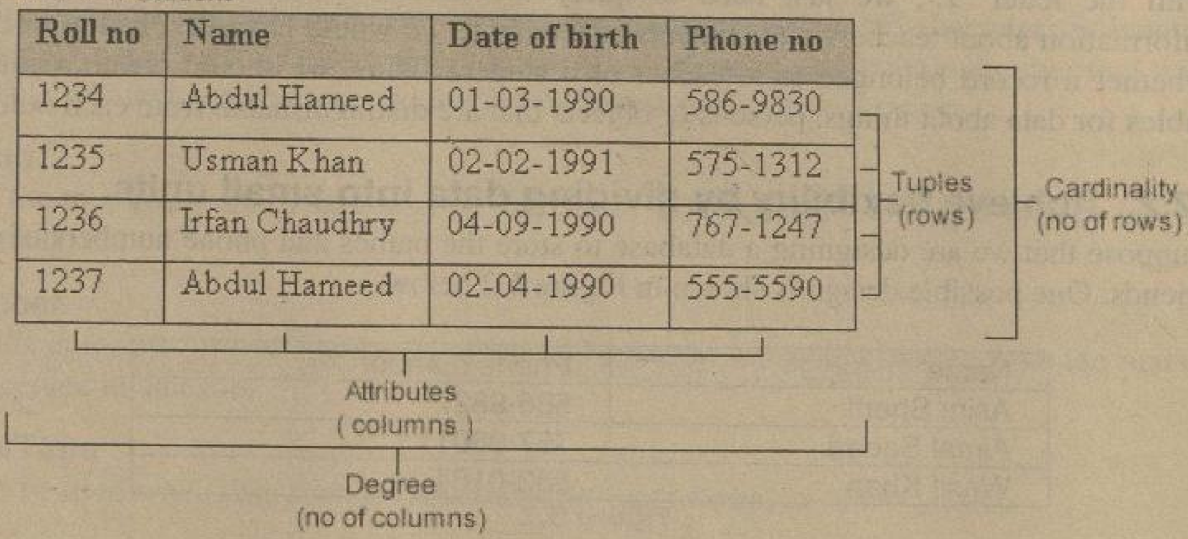
Relational Database

A relational database system is a system in which:

1. The user views all the data as being stored in the form of tables.
2. The operations that are provided by the relational system operate on tables only and produce as their output, data in form of tables. For example there will be operations that allow the user to extract all the rows of a table that meet a certain criteria. Note that the rows extracted can themselves be regarded as tables.

In chapter 3 we introduced the concepts of table, row, column, record and field. The relational model has its own terms for these concepts. The figure 5.1 below represents these terms graphically.

Relation Student



Roll no	Name	Date of birth	Phone no
1234	Abdul Hameed	01-03-1990	586-9830
1235	Usman Khan	02-02-1991	575-1312
1236	Irfan Chaudhry	04-09-1990	767-1247
1237	Abdul Hameed	02-04-1990	555-5590

Figure 5.1: Example of a relation (table).

Relation: A relation corresponds to the concept of a table.

Tuple: A tuple is the relational term representing a row of a table.

Cardinality: The number of tuples is called the cardinality.

Attribute: An attribute corresponds to a column of a table.

Degree: The number of attributes is called the degree.

The concept of a **relational model** was created in 1970 by Dr. E.F. Codd. Dr Codd based the relational model on rigorous mathematical principles and ideas from set theory and predicate logic. It is important to realise that the terms from the relational model that we presented above, have rigorous mathematical definitions. However, the non-mathematical definitions we have given are enough for the purpose of this course.

5.2 Database Design Principles

Now that you know how to create databases and add tables to them, it is time to study the various database design principles. These design principles tell you which tables you should add to the database and what the structure of each table should be. The number of tables and their structure depends on the nature of the data you want to insert and retrieve from the database. So a major part of designing a database consists of analyzing the structure and type of data that you are going to put into the database and anticipating the nature of the queries (Queries are explained in chapter 7) that are going to be made. You should follow the design principles given below to improve the performance of a database.

5.2.1 Use multiple tables.

A database can consist of as many tables as required. Putting the information about a particular subject in a separate table makes it easier to manage the data. For example in our **School** database we should have separate tables for storing information about students and teachers. If we are asked to produce a list of students whose names begin with the letter 'Z', we just need to query the **Student** table. If we had put the information about teachers and students in one table we would have first had to decide whether a record belonged to a teacher or a student. Thus, we should create separate tables for data about things, persons or objects that are distinguishable from each other.

5.2.2 Achieve flexibility by dividing data into small units.

Suppose that we are designing a database to store the names and phone numbers of our friends. One possible design is shown in Figure 5.2 below.

Name	Phone number
Asim Sharif	586-9827
Ajmal Saeed	757-9801
Wasif Khan	583-0184

Figure 5.2

Note that we have put the whole name into one field. If we are asked to sort our list in alphabetical order according to the Last Name of each person we will not be able to do it. The best we can do is to sort them alphabetically according to their first name. If we change the structure of our table so that the last names and first names are stored in separate fields, as shown in Figure 5.3, we will not have this problem.

First Name	Last Name	Phone Number
Asim	Sharif	586-9827
Ajmal	Saeed	757-9801
Wasif	Khan	583-0184

Figure 5.3

Now we can simply sort the table according to the First Name or the Last Name. Therefore, data should be stored in the smallest discrete units, which we want to retrieve or manipulate.

5.2.3 Choose the right Data type for fields

When we create a table we have to declare the name of the fields along with their data type. For example, in the **Student** database we defined the data type of the Roll number field as **Number**. Let us look at some of the choices Access provides for data types (see Figure 5.4) and the cases in which they should be used.

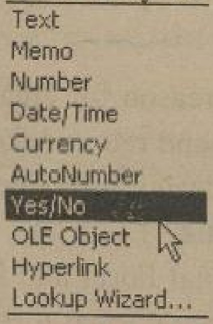


Figure 5.4:Data Type selection menu

Text

Use this data type when the field will contain any text up to 255 characters in length. Numbers that will not be used in numerical calculations, and certain number that will contain letters, hyphens etc should also be stored under this data type.

Memo

If the text you want to save will be more than 255 characters in length use the memo data type.

Number

This is meant for true numbers, which will be used in calculations such as the marks received in an exam.

Date/Time

To be used when data such as birth dates or time of day has to be stored.

Currency

To store data representing quantities in Rupees or other currencies such as payments, fees etc use this type.

AutoNumber

When a field is declared to have AutoNumber as its data type it will be assigned a number automatically each time a new record is added. For example, if there is a serial number field in the table you can declare it to be of type AutoNumber and the serial numbers will be assigned automatically each time you add a new record.

The remaining data types are only used in special cases and shall not be discussed here. Choosing the right data type saves space and speeds up the data management and processing tasks.

5.2.4 Set a Primary key

If there is a field (or a group of fields) in your table that will have a different value for each row in the table you can declare that field (or the group of fields) as a primary key (See section 3.2.3). Defining a primary key has the following advantages:

- The database management system will not allow duplicate entries in the primary key field
- Data processing tasks such as sorting and searching are speeded up.

5.2.5 Normalize Tables

As was pointed out in the first chapter the reason for creating computer databases was to provide an efficient and fast way to store and retrieve data. To ensure that we can use the database in an efficient manner certain design principles have to be followed. These design principles are called **Normal Forms**. There are five such principles usually called the First Normal Form, Second Normal Form and so on up to the Fifth Normal Form. Databases designed using the first three normal forms are said to be ***normalized***. We will only discuss the First Normal Form in this book.

First Normal Form (1NF)

If you look at the **Student** table of the school database, you will notice that only one phone number is given against each student's name. What if Usman Khan and Abdul Hameed each have one more phone installed at their homes? How would we enter the second phone number against their names in the database? One way to do this would be to add another column to the database. Our table would now look like Figure 5.5.

Roll no	Student Name	Father's Name	Date of Birth	Phone no 1	Phone no 2
1234	Abdul Hameed	Amir Hameed	3/19/90	586-9830	586-9865
1235	Usman Khan	Tariq Khan	2/25/91	575-1312	575-3409
1236	Irfan Chaudhry	Asim Chaudhry	9/4/90	767-1247	
1237	Abdul Hameed	Shahid Hameed	4/2/90	555-5590	

Figure 5.5: Sample Table

Repeating columns in this way has many problems:

- If you want to find out all the numbers starting with say the digits '586', you will have to search both the columns.
- What if the school also wants to add the mobile phone numbers of the students? This will require another change in the database design and since most students will have only one phone, a lot of columns will be blank which will waste space.
- Since the phone numbers are in two separate columns, we would not be able to use a primary key to ensure that duplicate numbers are not entered for the same student.

To conform to first normal form we have to redesign our database. Our database will now consist of two tables instead of one. Our first step is to create a new table named

Student Phone with the structure shown in Figure 5.6, with the combination of **Roll no** and **Phone no** fields set as the primary key.

Field Name	Data Type
Roll no	Number
Phone no	Text

Figure 5.6

Setting up a multiple field primary key is covered in Section 3.2.3. Once the table structure has been created copy data from the **Roll no** and **Phone no** fields of the Student table to this new table. Copying data is covered in section 4.5. Our new table, **Student Phone**, looks like this:

Roll no	Phone no
1234	586-9830
1235	575-1312
1235	575-1233
1236	767-1247
1237	555-5590

Figure 5.7: Student Phone

Next delete the **Phone no** column from the **Student** table. See section 4.3 for how to delete a column from a table. After removing the **Phone no** field, our **Student** table now has the form:

Roll no	Name	Father's Name	Date of birth
1234	Abdul Hameed	Amir Hameed	3/19/90
1235	Usman Khan	Tariq Khan	2/25/91
1236	Irfan Chaudhry	Asim Chaudhry	9/4/90
1237	Abdul Hameed	Shahid Hameed	4/2/90

Figure5.8

As you can see we have removed the field **phone** from the table **student**. The field **phone** is now part of another table called **Student Phone**. Note that we have put the roll no of each student along with the phone numbers in the new table, so that we know which phone number belongs to which student.

Let us see how this has solved the problems we listed earlier.

- If you want to find out all the numbers starting with say, the digits '586' only a single column has to be searched now. This increases the speed of the search process.
- You can add as many phone numbers per student as you like, by just adding another row with the student's roll no and the new phone number as its columns. No change in the design of the database is necessary.
- We can define the combination of the **Roll no** and **Phone no** columns as the primary key so that a duplicate phone number cannot be added for a student.

Our database has now been normalized to the First Normal Form (1NF).

Exercises

1. Normalize the **School** database that you have created in Access by following the instructions given in this chapter. Add the data of twenty of your classmates to the **Student** and **Student Phone** tables.

2. Add the tables with the following specifications to your *School* database using Access. Collect information from your school to add data to these tables.
3.

   a. **Table *Class***

   This table will list all the classes in the school. The field Class no is to be set as the primary key.

Field Name	Data Type
Class no	AutoNumber
Class Title	Text
Description	Text

Sample data for this table is given below.

Class no	Class Title	Description
1	5A	Class 5 Section A. Current Room assigned is Room no 23
2	6B	Class 6th Section B. Current Room assigned is Room no 24

   b. **Table Subject**

   This table lists the subject being taught to various classes in the school. Set Subject no as the primary key for this table.

Field Name	Data Type
Subject no	AutoNumber
Subject Title	Text
Description	Text

Sample data for this table is given below.

Subject no	Subject Title	Description
1	English5	English language course for class 5.
2	Science6	Science course for class 6.

c. **Table Student Class**

This table is a list of roll numbers of students and the classes they are enrolled in at the moment. Set the field Roll no as the primary key for this table.

Field Name	Data Type
Roll no	Number
Class no	Number

Sample data is given below

Roll no	Class no
1234	2
1256	3

d. **Table Class Subject**

This table relates a class and the subjects that are taught to that class. The primary key for this table is a combination of both the fields in the table.

Field Name	Data Type
Class no	Number
Subject no	Number

Sample data is given below

Class no	Subject no
1	2
1	3

Chapter 6

Relationships

Overview

Databases store information about the real world. Relationships between objects in a database management systems mirror the relationships that exist in the real world. The concept of relationships in database management systems helps us to better model the real world. It makes databases more useful in solving information management problems that are faced by companies or individuals. This chapter presents a brief introduction to the relationship-entity model and how this model is implemented using Microsoft Access.

6.1 Entity – Relationship Model

To make the task of designing a database easier computer scientists first create a model of the real world problem and then covert that model into specifications that represent the logical structure of the desired database. The model is called the **Entity-Relationship** data model. It represents the real world by first dividing the available data into various objects called entities. These entities are then linked to each other by defining relationships among them.

6.1.1 Entity

In a school a student represents a unique person. Similarly the subject of mathematics is different from all other subjects. An ***entity*** is a thing or an object in the real world that is distinguishable from all other objects. An entity has a set of properties and the values of some of these properties may uniquely identify an entity. For example the roll number of a student in school separates him from all other students.

6.1.2 Relationship

Entities have a set of properties associated with them. Each student has a set of properties e.g. **Roll no**, **Name**, **Father's Name** and **Date of birth**. When we create tables associated with entities, properties become fields of these tables. Phone numbers represent another entity in our school database. Each student has one or more phone numbers associated with him. We call such an association a ***relationship***. This relationship is represented graphically in Figure 6.1.

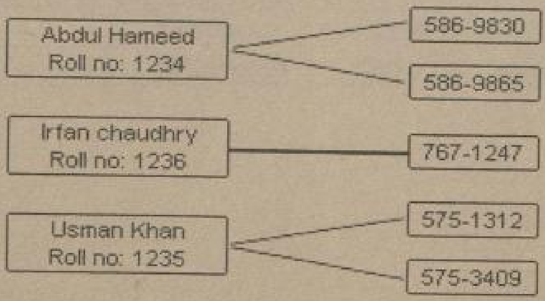


Figure 6.1: One-to-Many Relationship

The lines in the above diagram represent the relationships between entities. Note that each entity on the left has one or more lines connected to it while an entity on the right may have only one line connected to it. As you can see each student can be related to many phone but each phone is related to only one student, such a relationship is known as a **One-to-Many relationship**.

Now suppose that two brothers 'Usman Khan' and 'Hassan Khan' are in the same school. We get the relationship diagram (see Figure 6.2) for this case.

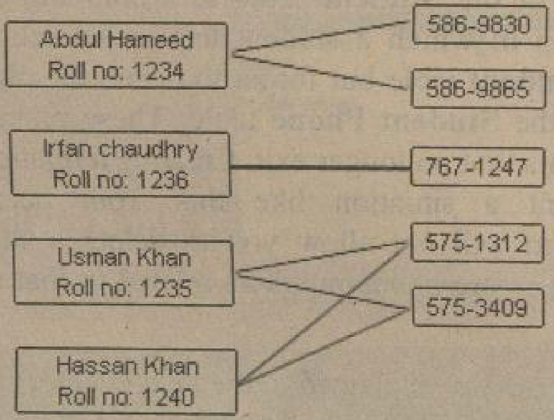

Abdul Hameed
Roll no: 1234
586-9830
586-9865
Irfan chaudhry
Roll no: 1236
767-1247
Usman Khan
Roll no: 1235
575-1312
575-3409
Hassan Khan
Roll no: 1240

Figure 6.2: Many-to-Many Relationship

Note that in this case each entity on the left as well as each entity on the right may have multiple lines connected to them. This represents the fact that each student may be related to many phone and each phone may be related multiple students, such a relationship is known as a **Many-to-Many relationship**.

The last kind of relationship we will study is called a **One-to-One relationship**. As an example (see Figure 6.3) consider the relationship between countries and their capitals.

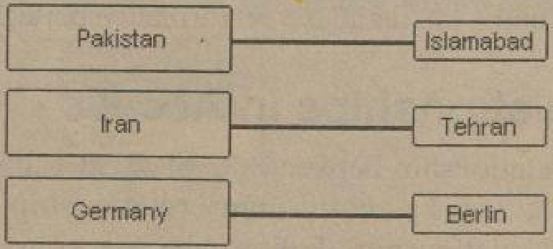

Pakistan
Islamabad
Iran
Tehran
Germany
Berlin

Figure 6.3: One-to-One Relationship

As you can see in the above diagram an entity on the left may be related to only one entity on the right, such relationships are known as **One-to-One relationships**.

6.2 Advantages of defining Relationships

When we implement a database the entities in the entity-relationship model become our tables in the database. We can then create relationships between these tables, just as we created relationships between entities in our model. Defining relationships has the following advantages:

- Referential integrity relationships between tables can be defined. Referential integrity ensures that the relationships between records in related tables remain consistent. For example if the **Student** and **Student Phone** tables are related, referential integrity ensures that every **Student Phone** record has a corresponding record in the **Student** table or in other words if there is a phone number present in the **Student Phone** table then we are assured of the fact that there is a student in the **Student** table to whom this number belongs. Now consider a situation in which a student leaves the school and you delete his record from the **Student** table but forget to delete the phone number records of that student from the **Student Phone** table. These phone number records will now refer to students that no longer exit. Creating a relationship between the two tables can prevent a situation like this from occurring. The database management system will not allow you to delete a student record from the **Student** table if there are phone numbers related to that student in the **Student Phone** table.
- When queries (Queries are explained in the next chapter) are made over multiple tables, the related tables will be joined automatically.
- Sorting and searching records according to some values in a specific field are fundamental operations provided by database management systems. To speed up these operations you can define ***indexes*** on fields of the table. When you index a field the database management system creates an ***index file*** associated with that field. The database then uses this file to speeds up sorting and searching operations based on this field. However indexes can slow down data entry and editing a little since each time you add or change data the index files must be updated also

  When you define relationships some indexes are created automatically in the related tables to make the database performance better.

6.3 Defining Relationships in Access

We will now define a relationship between the **Student table** and the **Student Phone** table in Access. Note that this is a **one-to-many relationship**, with the Student table on the "one" side of the relationship. Follow the steps given below to define the relationship.

- Close all open tables.
- Click on **Tools** in the main menu and then choose **Relationships**. The **Relationships** window will appear with the **Show Table** dialog box as shown in Figure 6.4.

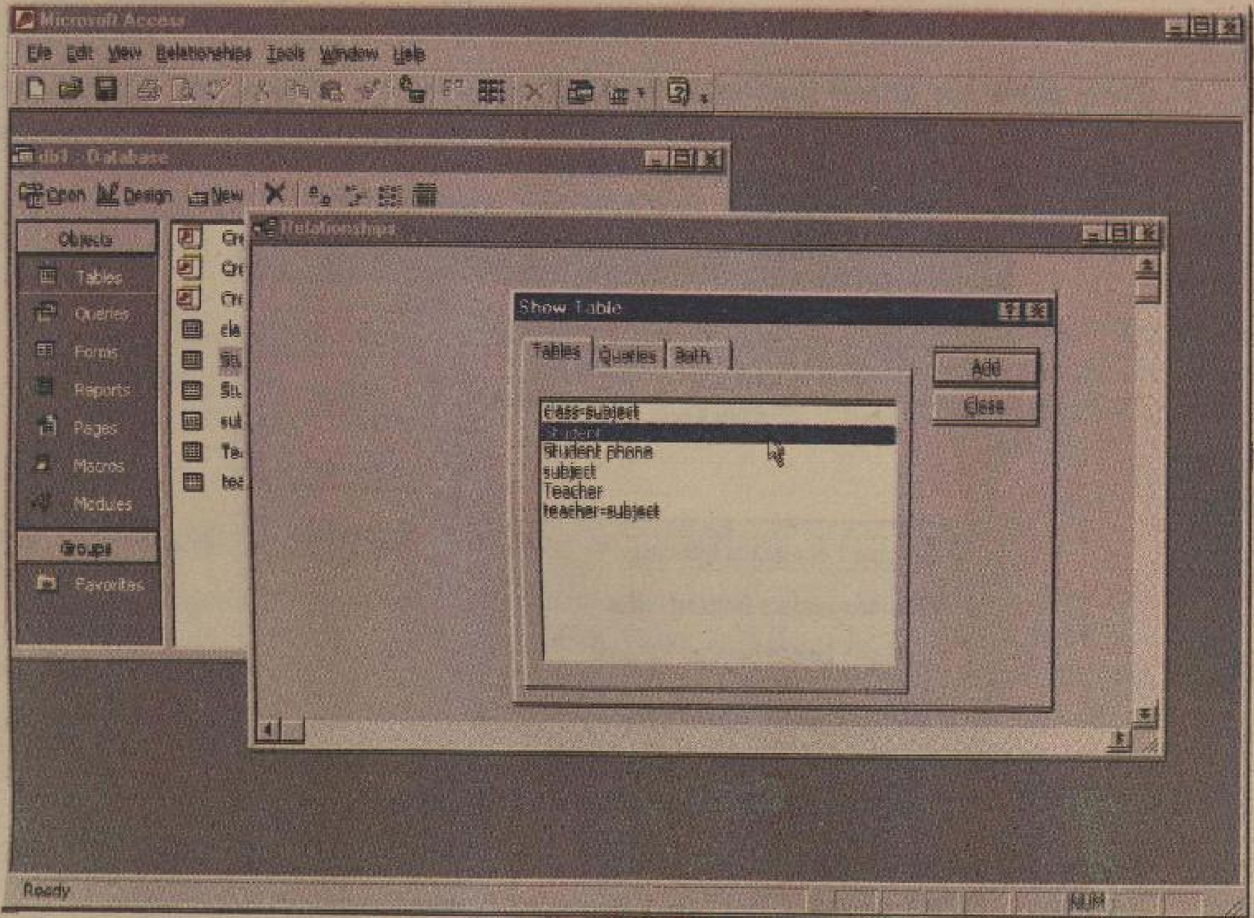

Microsoft Access
File Edit View Relationships Tools Window Help
db1 : Database
Open Design New
Objects
Tables
Queries
Forms
Reports
Pages
Macros
Modules
Groups
Favorites
Relationships
Show Table
Tables Queries Both
Add
Close
class-subject
Student
Student phone
subject
Teacher
teacher-subject
Ready
NUM

Figure 6.4: Show Table Window

If the **Show Table** dialog box does not appear click on **Relationships** in the main menu and choose **Show Table**. Double click on the **Student** and **Student Phone** table names this will add the tables to the relationship window as shown in Figure 6.5.

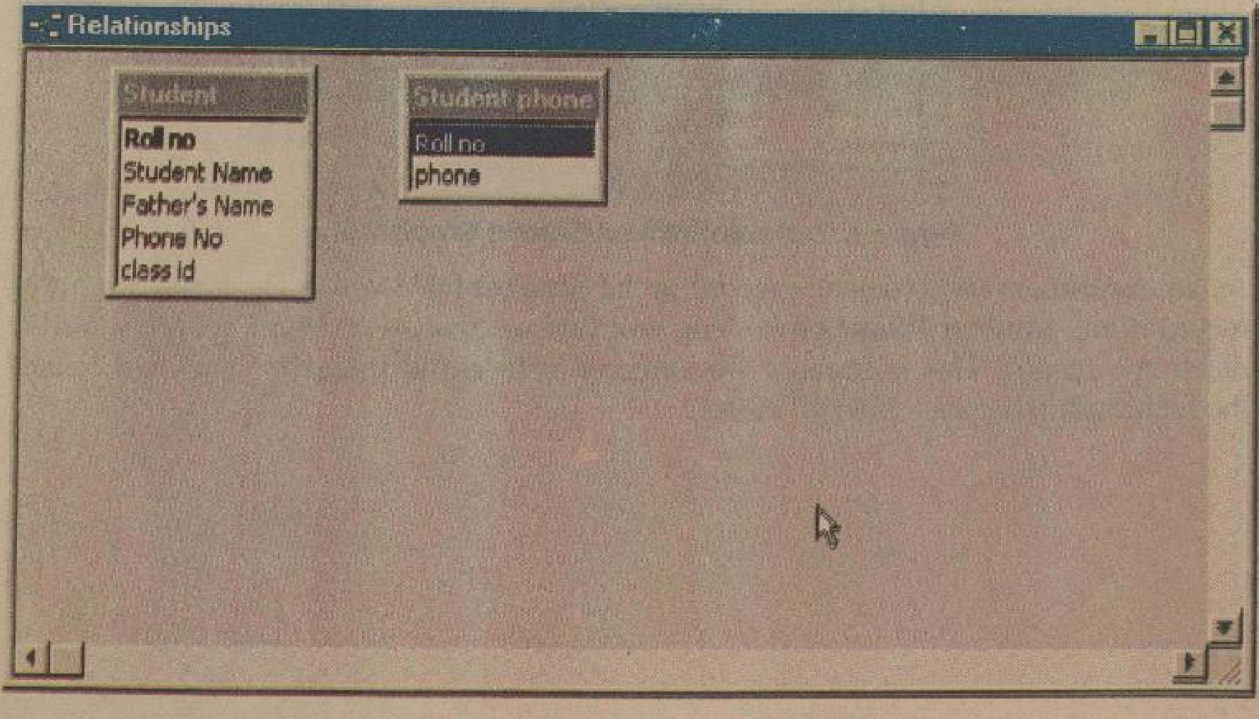
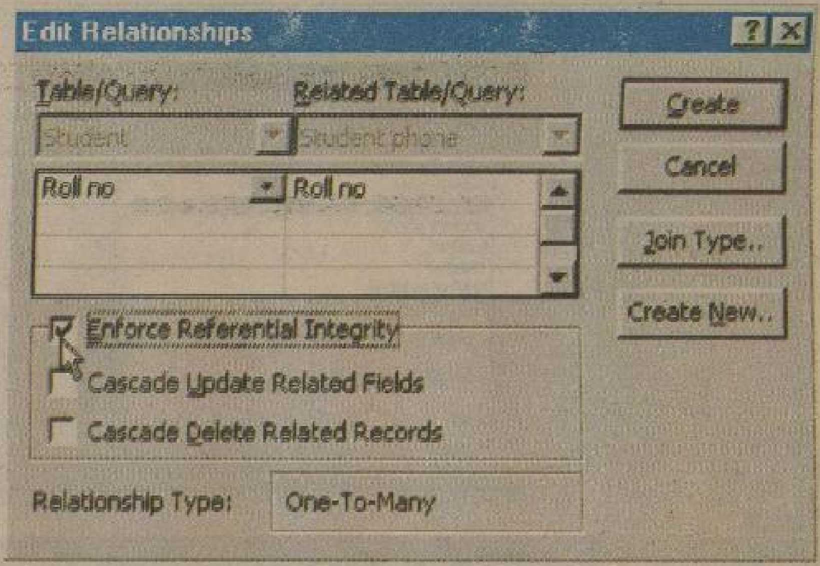

Relationships
Student
Roll no
Student Name
Father's Name
Phone No
class id
Student phone
Roll no
phone

Figure 6.5: Relationships Window

- Now move the mouse pointer to the **Roll no** field (the primary key field on the "one" side) and drag it to the **Roll no** field in the **Student Phone** table. Access will display a dialog box (see Figure 6.6).



Table/Query: Student

Related Table/Query: Student phone

Table/Query	Related Table/Query
Roll no	Roll no

☑ Enforce Referential Integrity

☐ Cascade Update Related Fields

☐ Cascade Delete Related Records

Relationship Type: One-To-Many

Figure 6.6: Edit Relationships Dialog Box

- Select the **Enforce Referential Integrity** box to enforce **referential integrity** between the two tables. Click on **Create** to create the relationship. Access will show the relationship between the two tables as a join line connecting the related fields (see Fig. 6.7). The symbol **1** near the Student table indicates that this is the table on the **one** side of the relationship and the infinity symbol ∞ indicates the table on the **many** side. Close the relationship window and save the relationship.

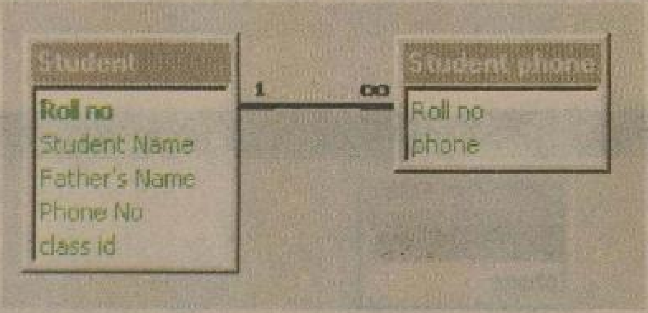


Figure 6.7: Relationships Shown Graphically

As an experiment try to delete a record in the **Student** table which has a corresponding record in the **Student Phone** table. You will find out that Access will not allow you to delete the record. This is because we enforced **referential integrity** when we created the relationship between the two tables.

Chapter 7

Creating Simple Queries

One of the main reasons we use databases to store data is that it allows us to perform queries on the data. Queries allow us to extract specific information from all the data stored in the database. We can also change data in the database that fulfills certain criteria.

7.1 Creating Queries from Scratch

You can create several types of queries in access. We will only study the most commonly type of query used, the ***Select Query***. It derives its name from the SQL **Select statement**. We will use the Access GUI to create queries, but you should know that in the background Access uses the SQL language to query the database. To get you started we will first create a query that extracts data from the **Student** table in our School database. The Student table has four fields in it **Roll no**, **Student name**, **Father's name**, **Date of Birth**. Our query should perform the following tasks:

- Extract the data in the **Roll no**, **Student name** and **Date of Birth** fields of all the records. Note that we don't want the **Father's name** field data to be extracted from the table.
- Display this data so that student names are in alphabetical order.

Here are the steps to do this:

- In the database window (see fig 7.1) click on the Queries tab.

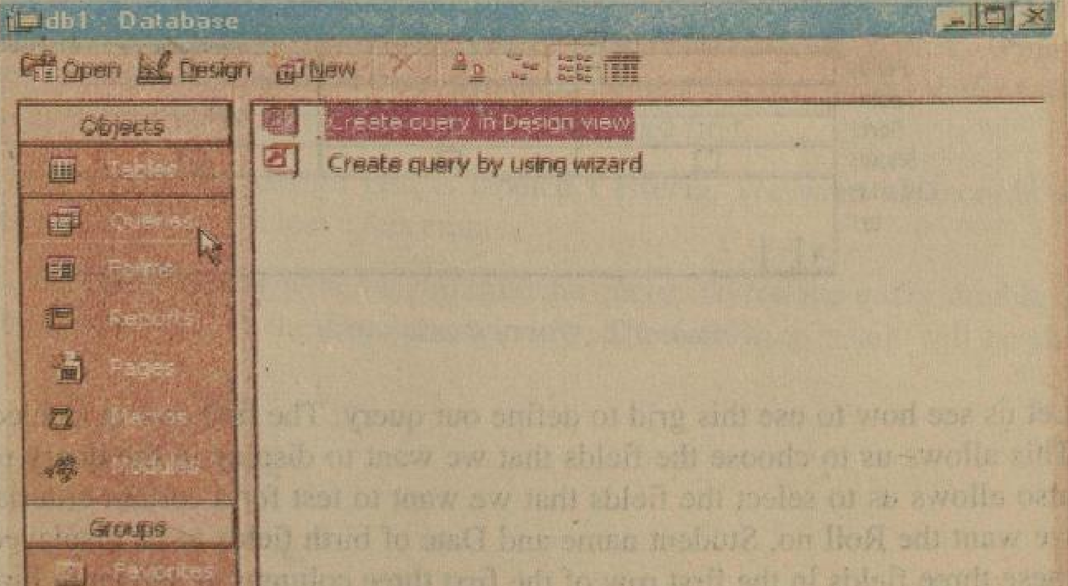

db1 : Database
Open
Design
New
Objects
Tables
Queries
Forms
Reports
Pages
Macros
Modules
Groups
Favorites
Create query in Design view
Create query by using wizard

Figure 7.1: Database Window with Queries Menu

- Double click on the **Create query in Design view** button. The **Select Query** window will open with the **Show table** dialog box (see Figure 7.2).

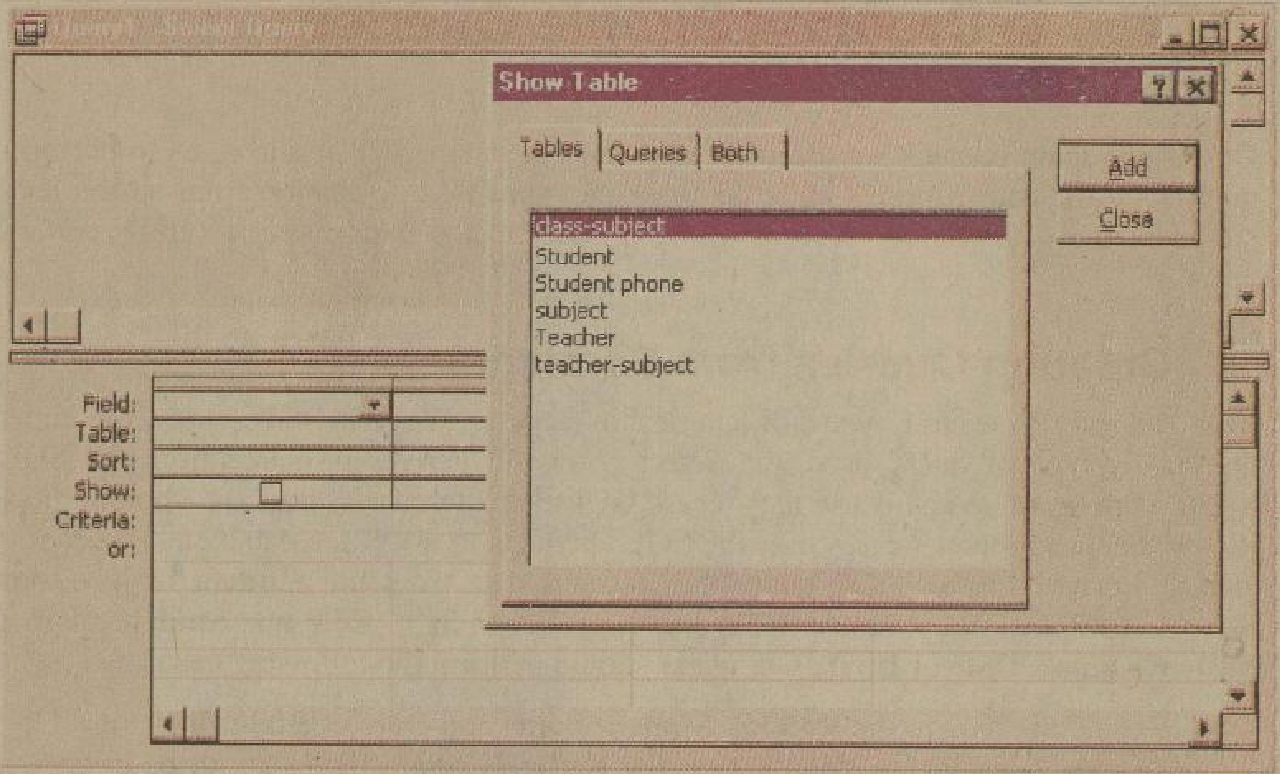

Show Table
Tables
Queries
Both
Add
Close
class-subject
Student
Student phone
subject
Teacher
teacher-subject
Field:
Table:
Sort:
Show:
Criteria:
or:

Figure 7.2: Show Table Dialog Box

- Queries are made on the data stored in tables. The show table allows us to select the tables we want to perform the query on. Double click on the **Student** table name to include it in the query window. **Close** the **Show Table** dialog box. The Query window will have the query design grid shown in Figure 7.3 in its lower half

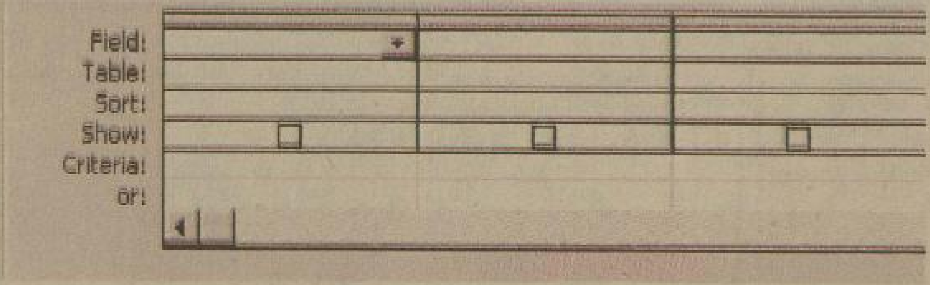
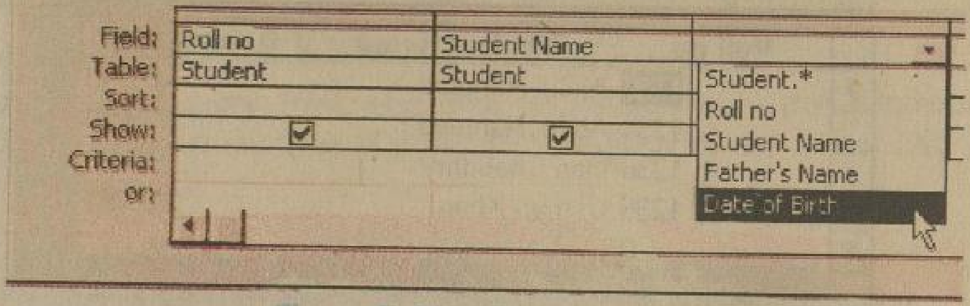

Field:
Table:
Sort:
Show:
Criteria:
or:

Figure 7.3: Query Design grid

- Let us see how to use this grid to define our query. The first row is labeled **Field**, This allows us to choose the fields that we want to display in the query result. It also allows us to select the fields that we want to test for a certain criteria. Since we want the Roll no, Student name and **Date of birth** fields to be displayed select these three fields in the first row of the **first three columns** as shown in Figure 7.4 below

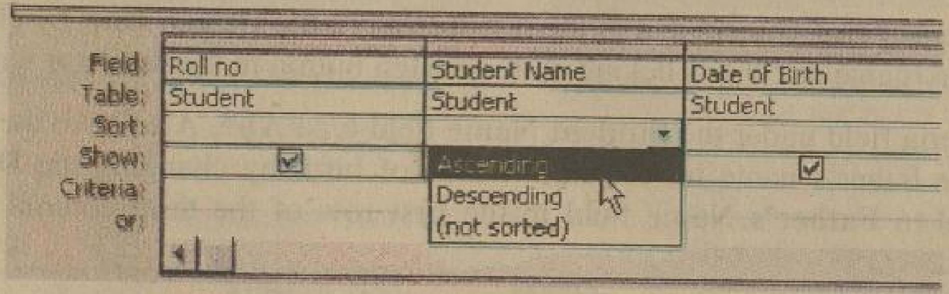


Field:	Roll no	Student Name	
Table:	Student	Student	
Sort:			
Show:	☑	☑	
Criteria:			
or:			

Figure 7.4: Defining a query

Note that Access automatically fills in the values in the second row labeled **Table**. This is because there is only one table to choose from. If we had selected more than one table for the query and a field name we were selecting was present in more than one table, we would have had to select the values in the **Table** row as well.

- The next row in the design grid is the **Sort** row. Since we want the records to be sorted according to the **Student Name** field in alphabetical order choose **Ascending** in the **Student Name** column (see Figure 7.5)

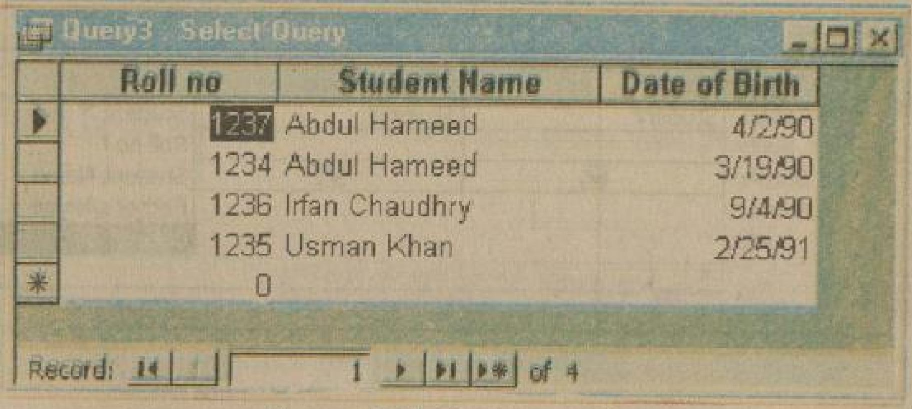


Field:	Roll no	Student Name	Date of Birth
Table:	Student	Student	Student
Sort:			
Show:	☑		☑
Criteria:			
or:			

Figure 7.5: Sorting according to a field

If we had chosen **Descending** the students names would have been displayed in the reverse alphabetical order, that is names starting with a Z would be displayed before names starting with an A. Note that we have left the **Sort** row of the **Roll no** and **Date of Birth** fields empty. If we choose to sort the **Date of Birth** field in ascending order also we would be performing a **sort within a sort**. What this means is that if two students have the same name the name of the student whose birthday occurs on an earlier date will be displayed first.

- The next row in the design grid is labeled **Criteria**. We want all records to be displayed therefore we leave this empty.
- Close the Select Query window and save the query. To run the query double click on the query name in the Database window. The following result will be shown


Query3 : Select Query
Record: 1 of 4

Roll no	Student Name	Date of Birth
1237	Abdul Hameed	4/2/90
1234	Abdul Hameed	3/19/90
1236	Irfan Chaudhry	9/4/90
1235	Usman Khan	2/25/91
0		

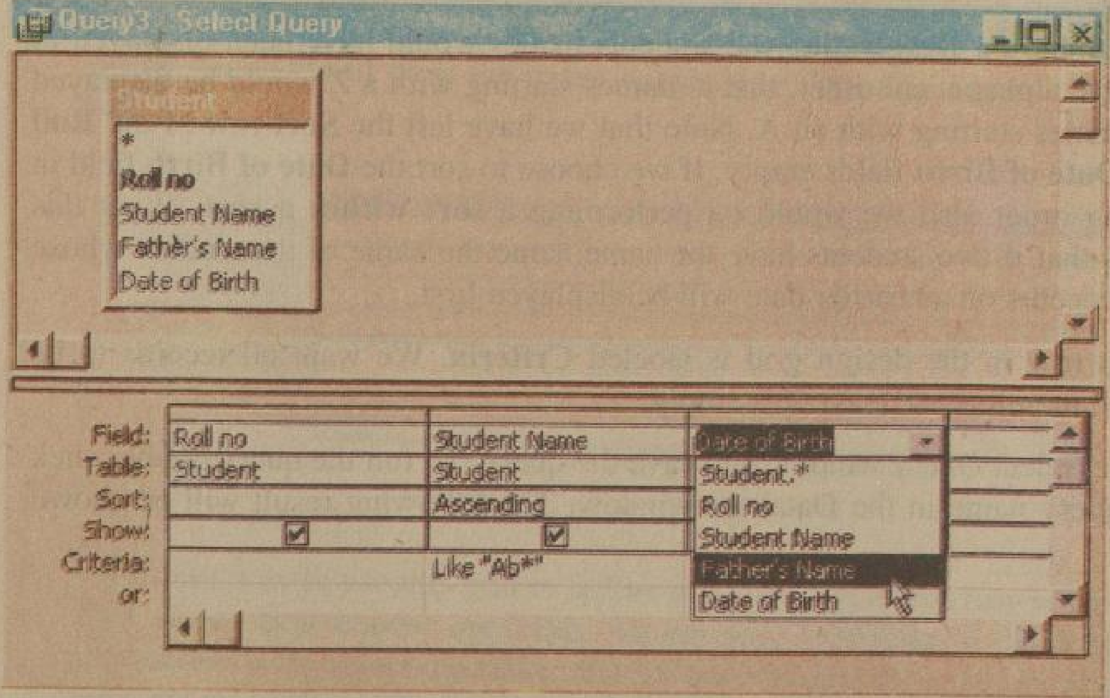
Figure 7.6: Query results

As you can see the records have been listed in the ascending order, and the **Father's name** field has not been displayed.

7.2 Queries based on Criteria

What if we wanted to list only certain rows that fulfilled some criteria? For example let us create a query that displays the **Roll no, Student Name** and **Father's name** fields of records of students whose names begin with the letters '**Ab**'.

- Open the query that you created in the last section in the **Design view** by clicking on the query name and then clicking on the design button in the database window.
- In the criteria field under the **Student Name** field type **Ab***. Also have the query display the father's name instead of the date of birth by changing the **Date of Birth** field to **Father's Name** field in the first row of the third column of the design grid.

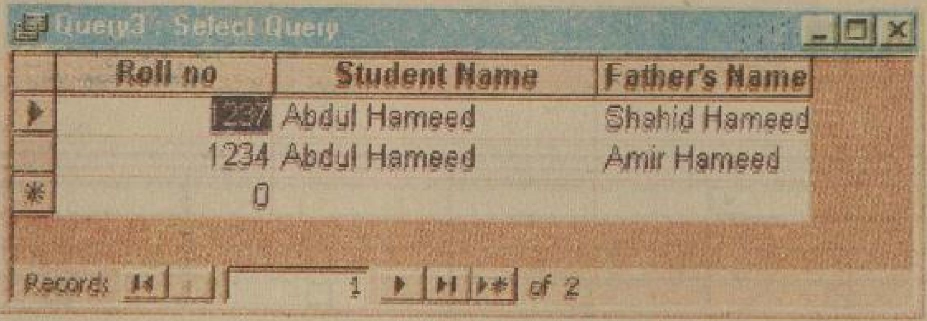

Query3 : Select Query
Student
*
Roll no
Student Name
Father's Name
Date of Birth
Student.*
Roll no
Student Name
Father's Name
Date of Birth

Field:	Roll no	Student Name	Date of Birth
Table:	Student	Student	
Sort:		Ascending	
Show:	☑	☑	
Criteria:		Like "Ab*"	
or:			

Figure 7.6: Defining Criteria Query

Notice that the letters **Ab*** you typed in the criteria field have changed to **Like "Ab"**. This tells Access to select only those records in which the student name begins with the letters **Ab**.

- Close and save the query. Now when you run the query only roll numbers, names and father's name of students whose name begin with the letters 'Ab' will be displayed, Figure 7.7



Roll no	Student Name	Father's Name
237	Abdul Hameed	Shahid Hameed
1234	Abdul Hameed	Amir Hameed
0		

Figure 7.6: Query Results

7.3 Wildcards

Note that we used the symbol * in our criteria cell. This is an example of the use of a wildcard character. Some other wildcard characters and their use is shown below.

Operator	Operator Meaning	Example	Example Meaning
*	Any characters	(619)*	Any text that starts with (619), e.g., phone numbers
?	Any Single character	P?-100	Values that start with P Followed by any single Charater, followed by -100
>	Greater than	>5000	Values greater than 5,000
=	Equals	=smith	Enteries equal to smith

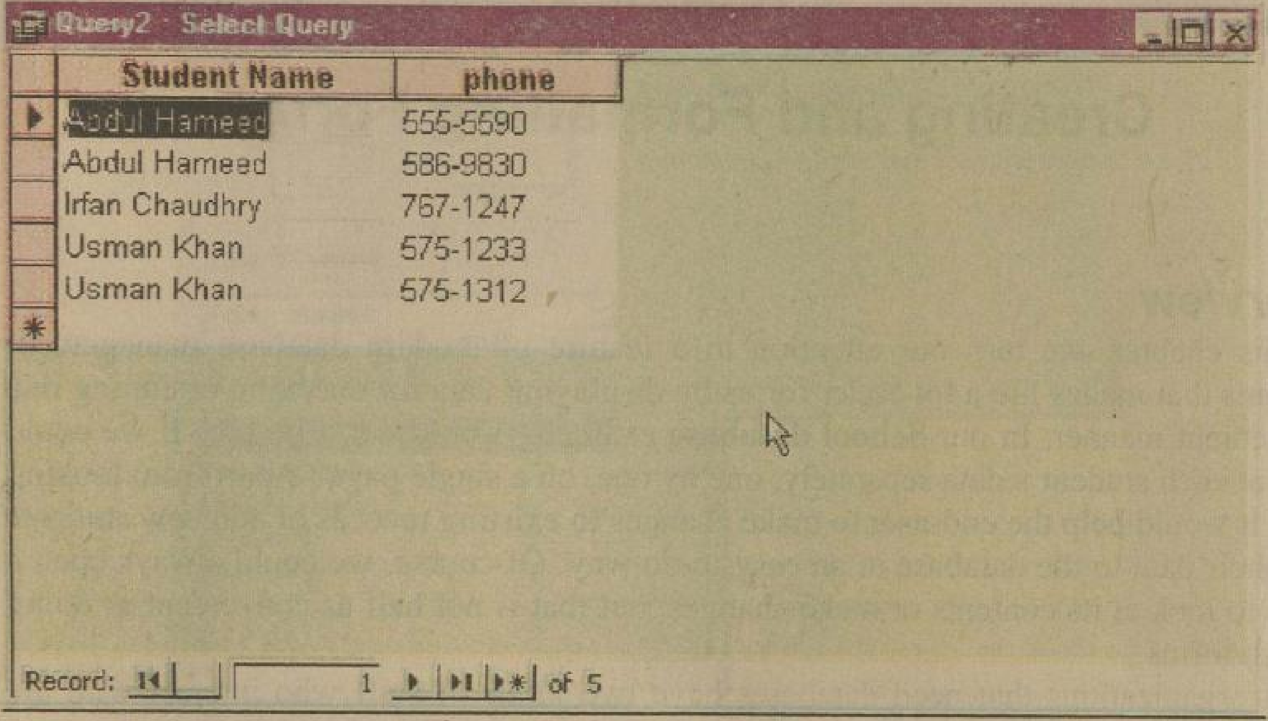
Figure 7.7: Wildcards

7.4 Joins: Querying Multiple tables

In the last section we queried the **Student** table to get a list of student names and the corresponding roll numbers. What if we wanted a list of student names along with their phone numbers? The names of the students are stored in the **Student** table while their phone numbers are stored in the **Student Phone** table. To create queries of this kind we need to include two tables in the query. Let us see how this is done.

- Create a new query and include both the **Student** and **Student Phone** table in it, see Figure 7.8. As you can see Access shows the relationship that we defined between the two tables in chapter 6
- Now select the appropriate values in the cells as shown in Figure 7.9
- Close and save the query. When you run the query the result shown in Figure 7.10 should be displayed

no field. Access looks at this information and automatically joins the two tables using the **Roll no** field.


Query2 : Select Query
Record: 1 of 5

Student Name	phone
Abdul Hameed	555-5590
Abdul Hameed	586-9830
Irfan Chaudhry	767-1247
Usman Khan	575-1233
Usman Khan	575-1312

Figure 7.10: Query Results

Chapter 8

Creating and Formatting Forms

Overview

In this chapter, we turn our attention to a feature of modern database management systems that makes life a lot easier for us by displaying data for checking or editing in a convenient manner. In our **School database** example, would not it be nice if we could look at each student's data separately, one by one, on a single page? Apart from looking neat, it would help the end-user to make changes to existing records or add new students and their data to the database in an easy-to-do way. Of course, we could always open a table to look at its contents or make changes, but that is not half as convenient as doing it with forms.

Most organizations that need databases have two kinds of staff who interact with the organization's database.. One type of staff is the technical people who act as database administrators. The other type is the non-technical staff, for example the school clerk who is responsible for entering data into the database and retrieving information from it. People of the second type are the end-users of the database. Remember that the end-users of the database may not be familiar with the concepts of databases and most probably lack training in the various techniques of interacting with a DBMS. Therefore, we need to provide these users with an easy way to use the database. That is where forms come in. ***Forms*** present a friendly interface to the non-technical users and allow these users to enter data into a database, edit data, and get information from a database without going into the technical details of a database.

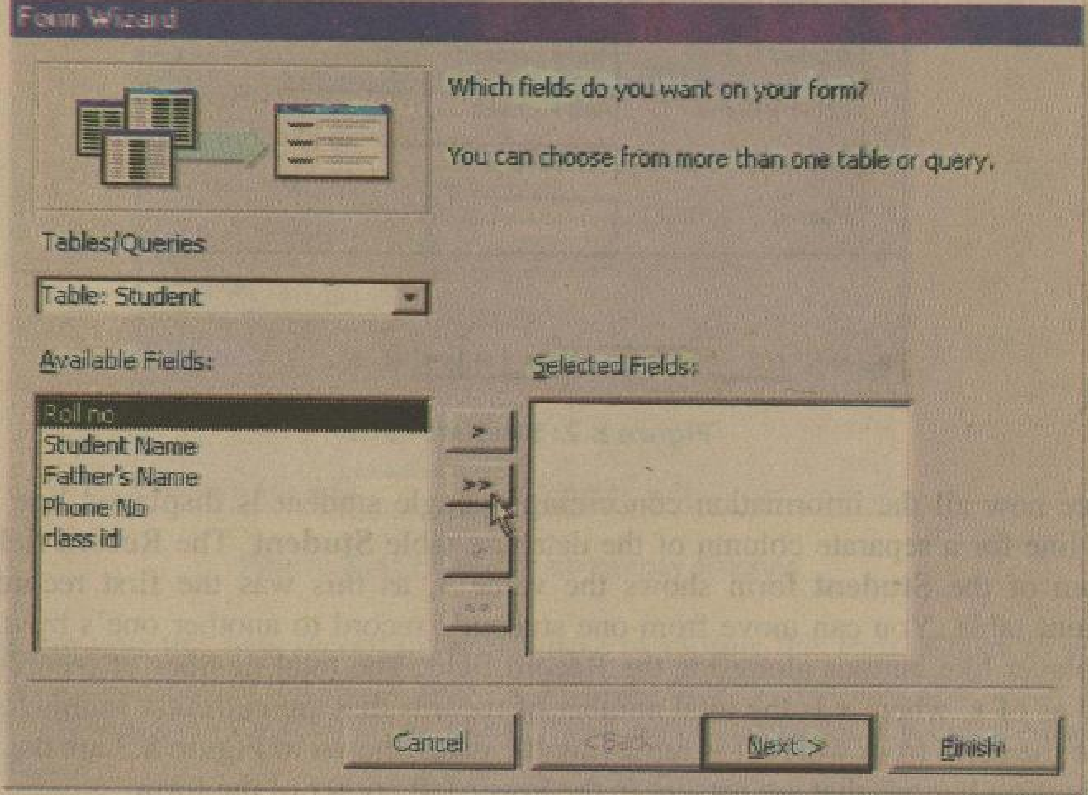
Forms are very easy to create. End users can themselves create forms or have them made by the DBA. In either case, the tables and their relationships etc. are all created by the DBA, and the end-user only has to deal with using these tables through forms and reports.

8.1 Creating a Single Table Form

The easiest way to create a form in **Access** is by using the **Form Wizard**. Let us create our first form that simply displays the records from the **Student** table.

- Open the **School database** and click on the **Forms** button under **Objects**.
- Double click on **Create Form by Using Wizard**. A screen by the name of **Form Wizard** opens which would let you select a table and the fields from that table that you want to include on your form. More than one table and even a query can be selected, but we are not going to do that right now. Just a single table will do.
- A drop-down list of all the tables present in the database is available under the caption **Tables/Queries,** see Figure 8.1. A table is already selected there by default and all its fields are written down in a bigger window underneath the table name. Select the table **Student** from the drop-down list if it in not already

selected there by default. This would make all its fields appear in the **Available Fields** window below.



Which fields do you want on your form?

You can choose from more than one table or query.

Tables/Queries

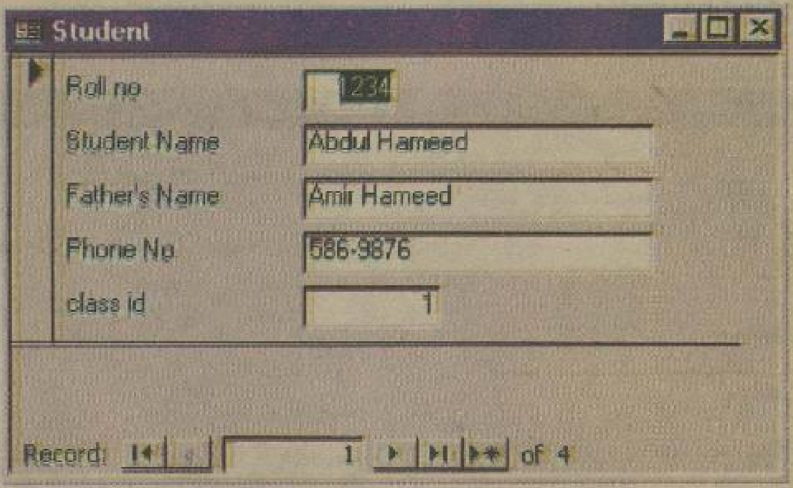
Table: Student

Available Fields:

Roll no
Student Name
Father's Name
Phone No
class id

Figure 8.1: Form Wizard

- Move all the fields in the **Available Fields** window to the **Selected Fields** window by clicking the >> button. Note that we can also select one or more fields by clicking on that field in the **Available Fields** window, and then clicking on the > button. Also, we can remove one or all selected fields by pressing on the < or << buttons respectively.
- Click **Next** at the bottom of this screen.
- One the next screen, the wizard asks you the **layout** in which you want your form to be presented. The available options are **Columnar**, **Tabular**, **Datasheet**, and **Justified**. These are different ways in which the fields can appear on your form. Select the choice **Columnar**. Click **Next** to go on to the next screen.
- The next screen of the Form Wizard lets you decide the **background** (*style*) that you want for your form. Choose **Standard** for now and move on by clicking **Next**.
- In the last step, give a name to your form. Note that the wizard displays a default name that is the name of the table from which this form is being created. Therefore, we let our form have the name **Student** too although we can change it if we want to. Note that a choice **Open the Form to View or Enter Information** is selected by default. Click **Finish** to complete creating your first form. Access will present you with the form shown in Figure 8.2:



Roll no 1234
Student Name Abdul Hameed
Father's Name Amir Hameed
Phone No 586-9876
class id 1

Figure 8.2: Student Form

Notice how all the information concerning a single student is displayed line by line, each line for a separate column of the database table **Student**. The Record field at the bottom of the **Student** form shows the value 1, as this was the first record of the **Student** table. You can move from one student's record to another one's by using the CD-player like buttons alongside the Record field. The total number of records is also listed as **of x**, where **x** is the total number of records that are currently in this form, and hence the table from which it is retrieving the data. Shown in Figure 8.3 are the **Record Navigation** buttons that are present in the lower-left corner of the form:

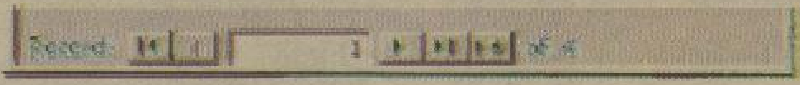


Figure 8.3: Record Navigation Toolbar

Click these buttons to move to different records. The Last Record button displays an empty record and is used to enter new records.

8.2 Editing Records through Forms

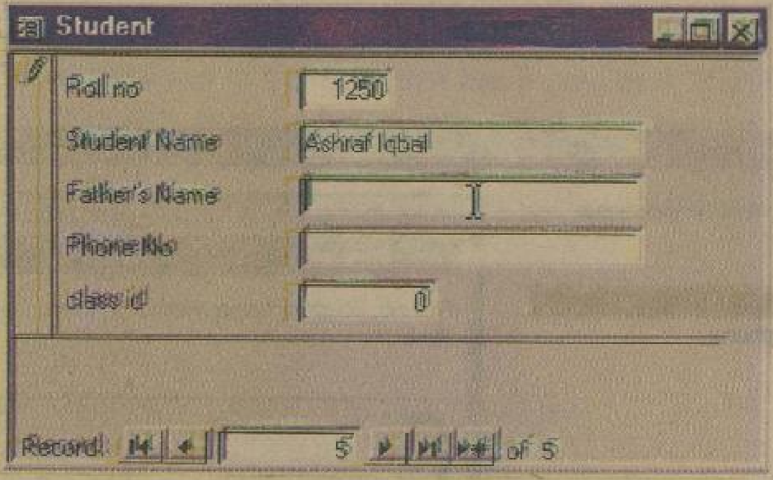
After you have familiarized yourself with this form by going from one record to another in it, it is time to edit some data. Go to record 1 in the form. Change the **Student Name** from Abdul Hameed to Muhammed Abdul Hameed and close the form. Now open the table ***Student*** and look at the first record. You will find that the name of the first student has indeed changed in the **Student** table.

8.3 Adding new records through Forms

Suppose another student has just been admitted to the school and we need to enter his information in the school's database. All we need to do is to:

- Open the form by opening the **School database** clicking on **Forms** under **Objects** and then double clicking on **Student**.

- Click on the **New Record** button to move to an empty record. Fill in the various fields with the data of the new student. An example is shown in Figure 8.4 below:



Roll no 1250

Student Name Ashraf Iqbal

Father's Name

Phone No

class id 0

Record: 5 of 5

Figure 8.4: Filling in an empty Record

- You can save this new record in the database by closing this form. Check to see if the new information is entered into the **Student** table by opening that table. You will find that the new record has indeed been added automatically to the **Student** table.

8.4 Creating a Form from two tables

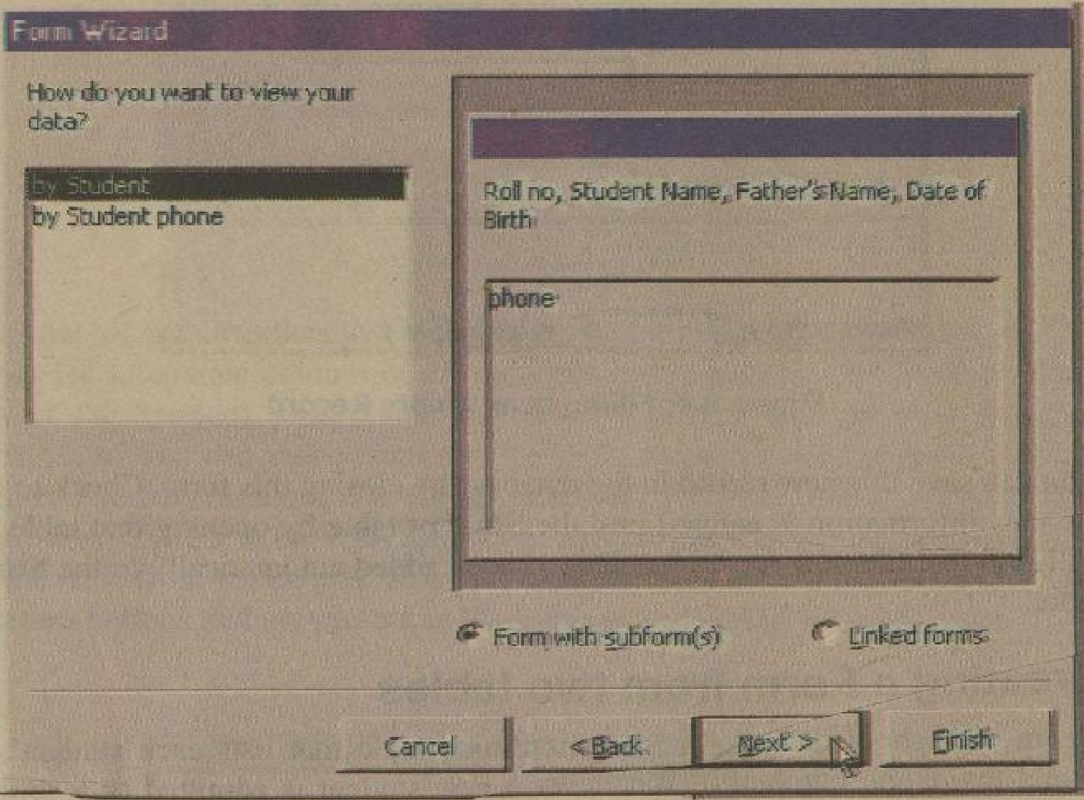
We have in our school database a **Student-Phone** table that lists each student's roll number and his phone number(s). Whenever a new student is admitted or a student's phone number is changed, we have to make changes in either this table or the **Student** table. We could have another single table form created for the **Student-Phone** table and work through that, but that would not be convenient. Moreover, it would be nice to be able to see a student's phone number(s) along with his other data from the **Student** table, all in one form. This way, the user can make any changes to either table through a single form.

The following steps take you through the process of creating a form, **StudentMulti**, out of the two tables **Student** and **Student-Phone**. The **StudentMulti** form would have a **subform** within it that displays the student's phone number(s).

- Open the database, click the **Forms** button in the **Database window**, double click **Create Form by Using Wizard**.
- In the first step of the Form Wizard, select the table **Student** from the drop-down list **Tables/Queries**. Move all the **Available Fields** to the **Selected Fields** by pressing the >> button. Do not click on Next yet.
- Select the table **Student-Phone** from the **Tables/Queries** drop-down list. Click on the field **Phone no** only in the **Available Fields** window and press > once to move only this field to the **Selected Fields** window. We could select the **Roll**

no field too from this table if we wanted to, but there is no need to do so as the **Roll no** field is already present from the **Student** table. Click **Next**.

- The next screen of the wizard asks **How do you want to View your Data?** By default, the value is **By Student** and the screen looks like:



How do you want to view your data?

Roll no, Student Name, Father's Name, Date of Birth

phone

● Form with subform(s) ○ Linked forms

Figure 8.5: View Data Choice

Note that the button Form with **Subform(s)** is selected on this screen by default. Click **Next**.

- Proceed as you did for the single table form, choosing a **layout** (choose Datasheet for now), a **style**, and a **name** (**StudentMulti**) for this form and the **subform** (the default **Student-Phone** is good enough). Click **Finish**.

The form opens with the first student's roll number, name, father's name, date of birth at the top and his phone number(s) below in another window, which is the window for the **subform**. You can check that two forms have been created with the names given by you in the previous step, by pressing on the **Forms** button in the **Database window**. By clicking on the **StudentMulti** form, you open the form that lists all the student information along with the phone numbers. The **Student-Phone subform** is part of the **StudentMulti** form and opening the **Student-Phone subform** alone would only list the phone numbers of students and will not be very useful.

Notice that the **form** and the **subform** within it have separate **navigation bars** so that you can edit data in either form.

8.5 Editing and Adding Data in Multi-Table Forms

We can change data for any student by using this form. For example, if somebody has a new phone connection, we can open the **StudentMulti** form and simply go to that student's record using the lower navigation buttons. We can then add a phone number in the **Student-Phone subform** in the field that has a zero in it. Similarly, we can edit any field visible on this form, or add a brand new record for a new student and all these changes would be stored in the database. We can check this by opening the **Student** table and the **Student-Phone** table and verify that the tables have been updated.

Exercises

1. Select different Layouts and see how the form looks.
2. In the View by table choice window, instead of choosing **by Student,** try choosing **by Student-Phone** and see the difference between the two.

Chapter 9

Creating and Formatting Reports

Overview

Reports retrieve data from the database and present it to the user in a formatted way. Here, we are not concerned with making changes to the database but only want to extract information from it in a presentable and convenient manner. In our **School** database example, the principal might want a list of all the students currently enrolled. Of course, we could just print the **Student** table, but then we will not be able to format the data, or leave out any unwanted fields. In addition, what if the principal also wants the phone number(s) of each student listed along with the other student data? Now the phone numbers are in another table altogether. How to combine the two? The report creating facility in Access accomplishes this task with ease, by using both tables to create a single report.

The difference between a form and a report is that the former is a way to organize data so that it can be easily modified, or that new data can be input easily into the database. On the other hand, reports are only printed outputs and do not allow the user to input any data through them.

9.1 Creating a Single Table Report

The easiest way to create a report in **Access** is by using the **Report Wizard**. Let us create our first report that simply displays the records from the **Student** table.

- Open the **School database** and click on the **Reports** button under **Objects**.

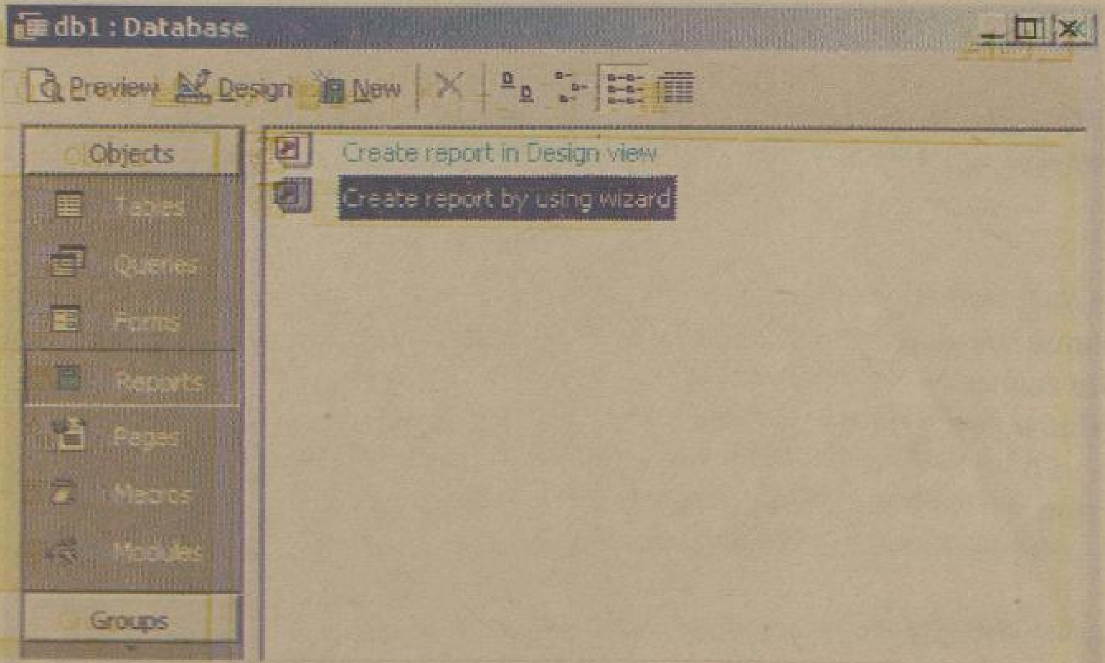
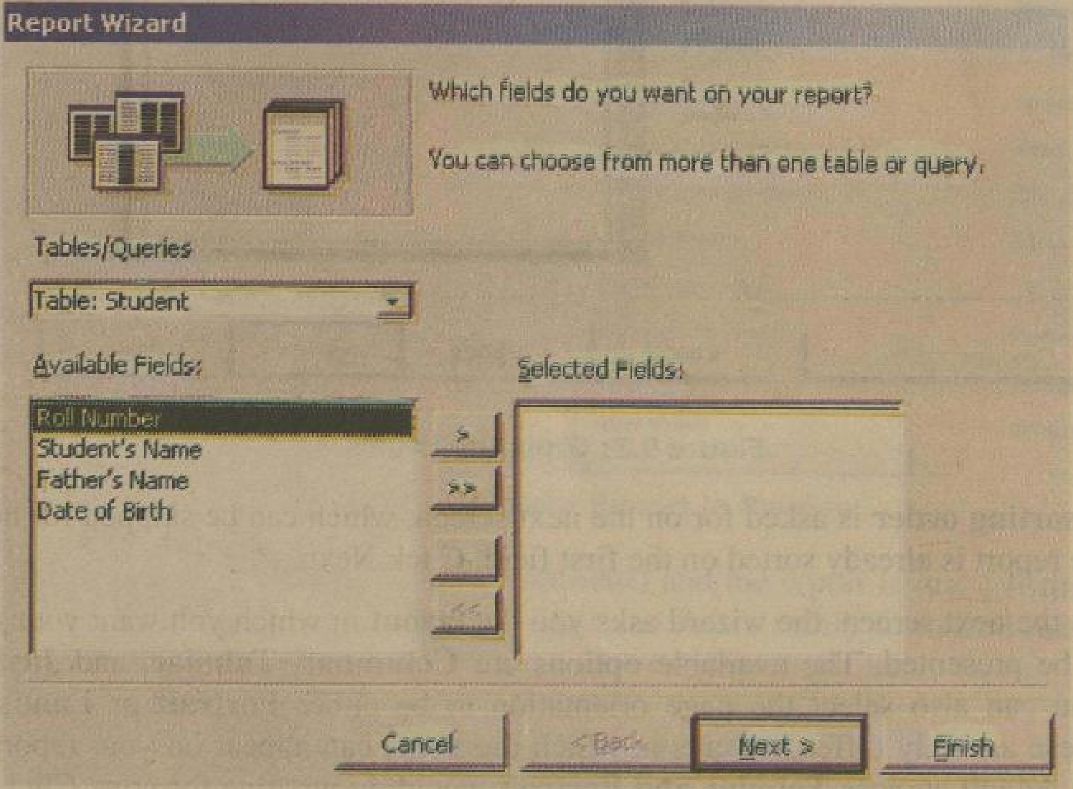

db1 : Database
Preview
Design
New
Objects
Tables
Queries
Forms
Reports
Pages
Macros
Modules
Groups
Create report in Design view
Create report by using wizard

Figure 9.1: Database Window with Reports Menu

- Double click on **Create Report by Using Wizard**. A screen by the name of **Report Wizard** opens which lets you select a table and the fields from that table that you want to include on your report. More than one table and even a query can be selected, but we are not going to do that right now. Just a single table will do.

- A drop-down list of all the tables present in the database is available under the caption **Tables/Queries**. A table is already selected there by default and all its fields are written down in a bigger window underneath the table name. Select the table **Student** from the drop-down list if it in not already selected there by default. This would make all its fields appear in the **Available Fields** window below.



Which fields do you want on your report?

You can choose from more than one table or query.

Tables/Queries: Table: Student

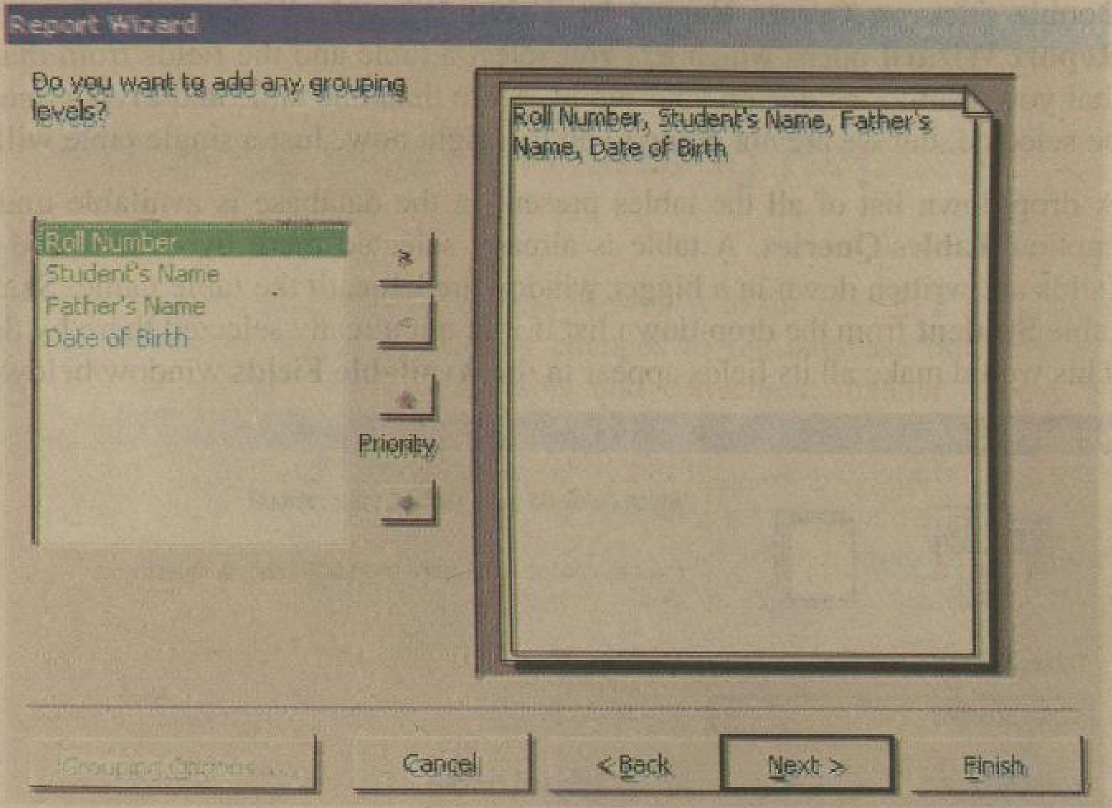
Available Fields:
Roll Number
Student's Name
Father's Name
Date of Birth

Selected Fields:

Figure 9.2: Choosing Fields for Report

- Move all the fields in **Available fields** of the **Student** table to **Selected Fields** by pressing the >> button to be displayed in our report. Note that we can also select one or more fields by clicking on that field in the left Available Fields window, and then clicking on the > button. Also, we can remove one or all selected fields by pressing on the < or << buttons respectively. Click Next at the bottom of this screen.

- The wizard asks if you want to add any grouping level, i.e. do you want to arrange the output so that records with similar data in one particular field are grouped together. If you do not select any grouping level, you will get the output in the same order of fields as they are in the original **Student** table. For now, do not select any grouping level and click **Next**.



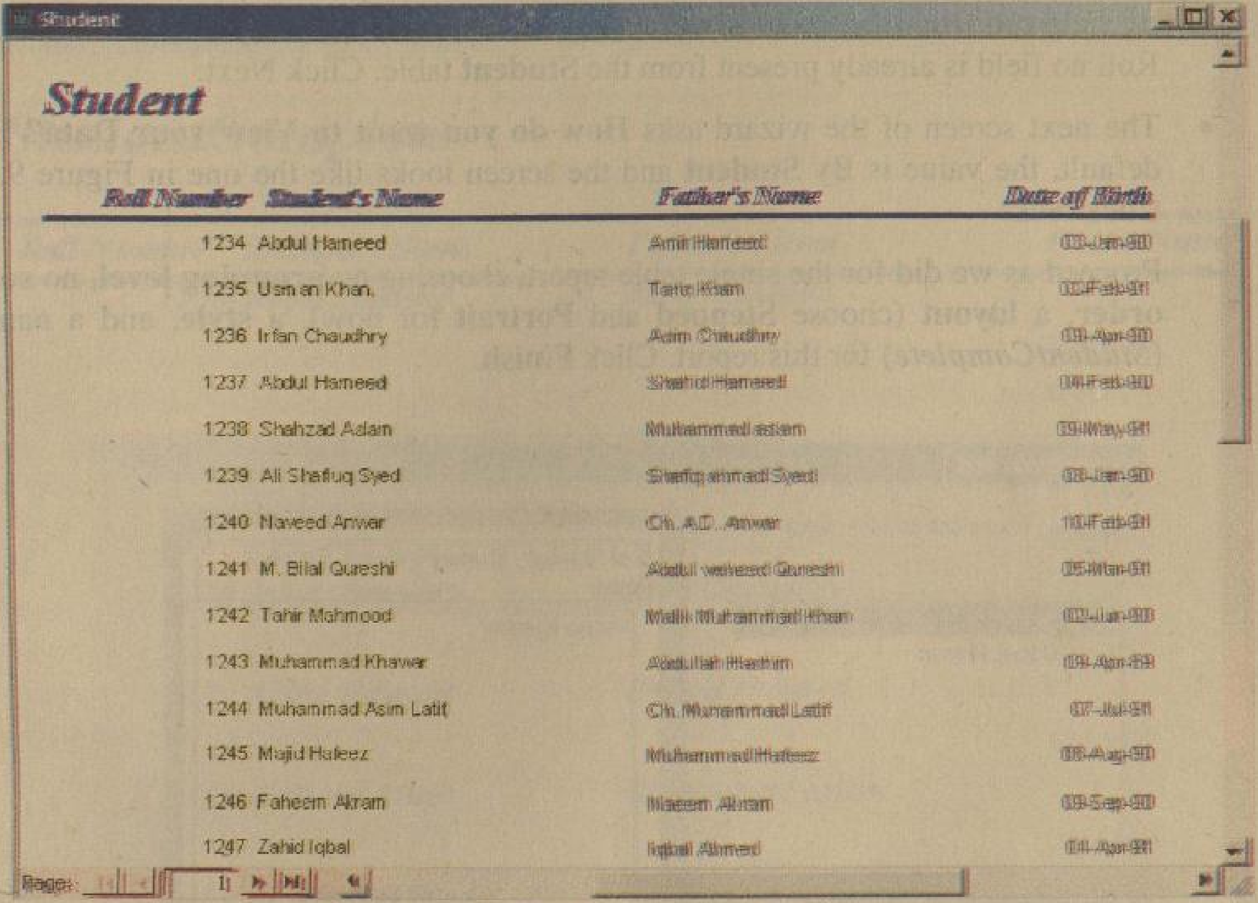
Do you want to add any grouping levels?

Roll Number, Student's Name, Father's Name, Date of Birth

Figure 9.3: Grouping Levels

- A **sorting order** is asked for on the next screen, which can be skipped for now as the report is already sorted on the first field. Click Next.
- On the next screen, the wizard asks you the **layout** in which you want your report to be presented. The available options are **Columnar**, **Tabular**, and **Justified**. You can also select the page orientation to be either **Portrait** or **Landscape**. These are only different ways in which the fields can appear on your report, and the default choices **Tabular and Portrait** would do just fine for now. Click Next to go on to the next screen.
- The next screen of the Report Wizard lets you decide the **background** (*style*) that you want for your report. Choose **Formal** for now and move on by clicking **Next**.
- In the last step, give a name to your report. Note that the wizard displays a default name, which is the name of the table from which this report is being created. Therefore, we let our report have the name **Student** too although we can change it if we want to. Note that a choice **Preview the Report** is selected by default. Let it be so and press **Finish** to complete creating your first report.

The report opens in a preview window and looks like this:



Student

Roll Number	Student's Name	Father's Name	Date of Birth
1234	Abdul Hameed	Amir Hameed	03-Jan-90
1235	Usman Khan.	Tariq Khan	02-Feb-91
1236	Irfan Chaudhry	Asim Chaudhry	03-Apr-90
1237	Abdul Hameed	Shahid Hameed	04-Feb-90
1238	Shahzad Aslam	Muhammad aslam	03-May-91
1239	Ali Shafiuq Syed	Shafiq ahmad Syed	03-Jan-90
1240	Naveed Anwar	Ch. A.D. Anwar	10-Feb-91
1241	M. Bilal Qureshi	Abdul waheed Qureshi	05-Mar-91
1242	Tahir Mahmood	Malik Muhammad Khan	02-Jun-90
1243	Muhammad Khawar	Abdullah Hashim	09-Apr-89
1244	Muhammad Asim Latif	Ch. Muhammad Latif	07-Jul-91
1245	Majid Hafeez	Muhammad Hafeez	08-Aug-90
1246	Faheem Akram	Naeem Akram	09-Sep-90
1247	Zahid Iqbal	Iqbal Ahmed	04-Apr-91

Figure 9.4: Student Report in Preview

The report's name appears as the title (**Student**) and the report is just a listing of each record from the **Student** table. The principal now has a nicely formatted list of all the students in the school.

9.2 Creating a Report from two tables

Consider the case in which we want each student's phone number instead of the date of birth in the report created in the last section. We have in our school database a **Student-Phone** table that lists each student's roll number and his phone number(s). We are now ready to use both **Student** and **Student-Phone** tables to create our next report.

The following steps take you through the process of creating a report, **StudentComplete**, out of the two tables **Student** and **Student-Phone**.

- Open the database, click the **Reports** button in the **Database window**, double click **Create Report by Using Wizard**.
- In the first step of the Report Wizard, select the table **Student** from the drop-down list **Tables/Queries**. Move all the **Available Fields** to the **Selected Fields** by pressing on the >> button. Do not click on Next yet.
- Select the table **Student-Phone** from the **Tables/Queries** drop-down list. Click on the field **Phone no** only in the **Available Fields** window and press > once to move only this field to the **Selected Fields** window. We could select the Roll

no field too from this table if we wanted to, but there is no need to do so as the Roll no field is already present from the **Student** table. Click Next.

- The next screen of the wizard asks **How do you want to View your Data?** By default, the value is **By Student** and the screen looks like the one in Figure 9.5. Click **Next**.
- Proceed as we did for the single table report, choosing no **grouping level**, no **sort order**, a **layout** (choose **Stepped** and **Portrait** for now), a **style**, and a name (***StudentComplete***) for this report. Click **Finish**.

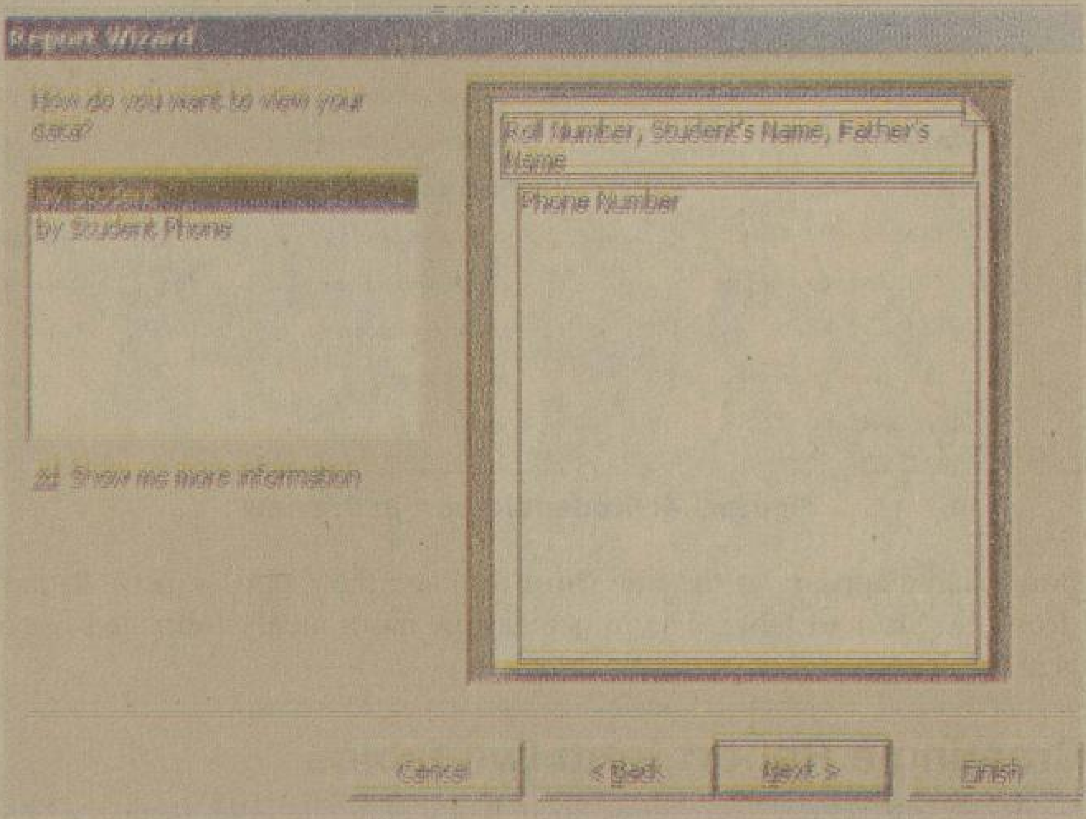
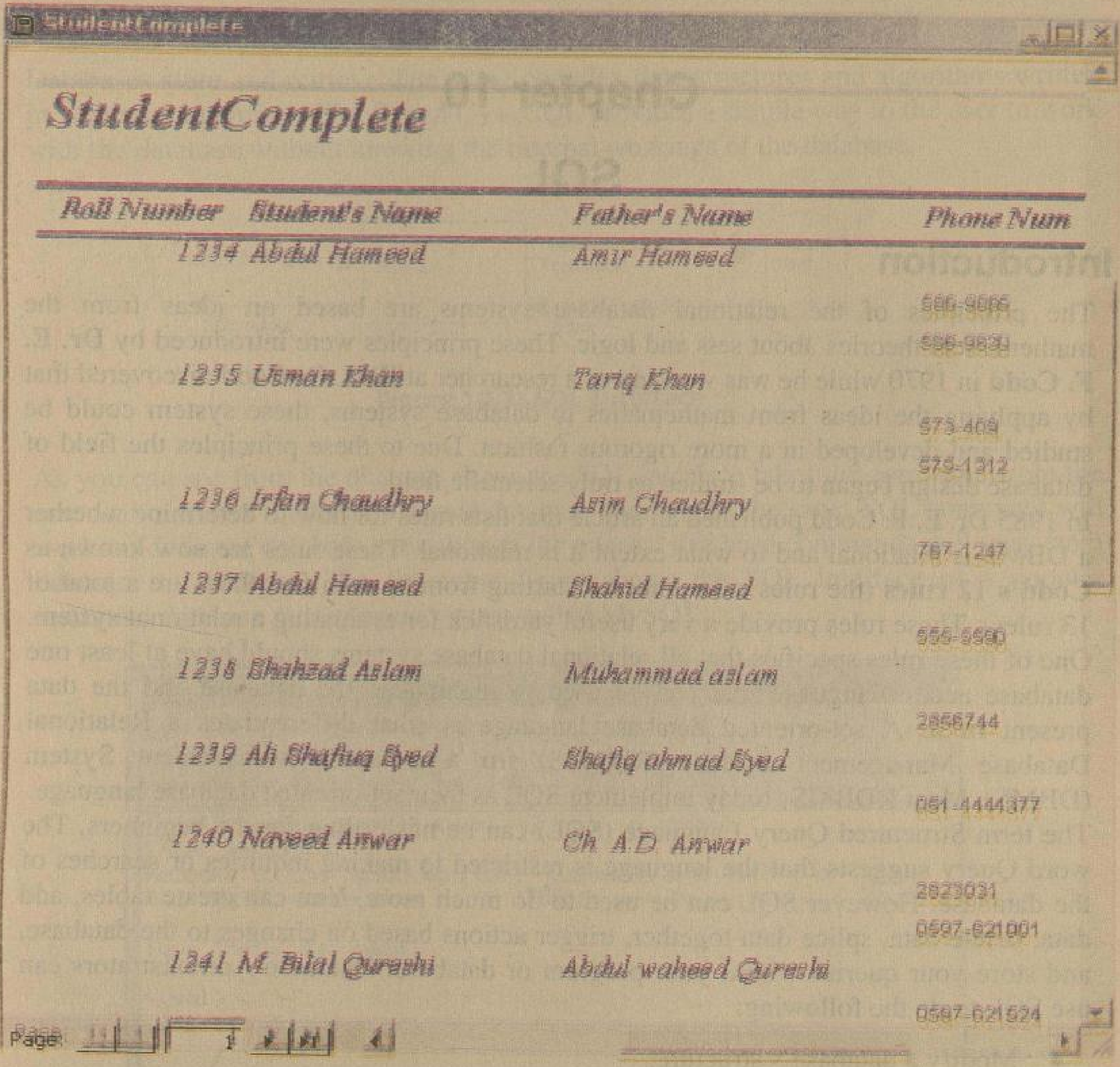

Report Wizard
How do you want to view your data?
by Student
by Student Phone
Show me more information
Roll Number, Student's Name, Father's Name
Phone Number
Cancel
< Back
Next >
Finish

Figure 9.5: View Data Choice

The new report opens and indeed has students' phone number(s) too in addition to all the data in the previous report, see Figure 9.6.



StudentComplete

Roll Number	Student's Name	Father's Name	Phone Num
1234	Abdul Hameed	Amir Hameed	
			586-9666
			586-9833
1235	Usman Khan	Tariq Khan	
			573-469
			575-1312
1236	Irfan Chaudhry	Asim Chaudhry	
			787-1247
1237	Abdul Hameed	Shahid Hameed	
			555-5580
1238	Shahzad Aslam	Muhammad aslam	
			2856744
1239	Ali Shafiuq Syed	Shafiq ahmad Syed	
			061-4444377
1240	Naveed Anwar	Ch. A.D. Anwar	
			2823031
			0597-621001
1241	M. Bilal Qureshi	Abdul waheed Qureshi	
			0597-621524

Figure 9.6: StudentComplete Report in Preview

Exercises:

1. Select different Layouts and see how the report looks.
2. In the View by table choice window, instead of choosing **by Student,** try choosing **by Student-Phone** and see the difference between the two.

Chapter 10

SQL

Introduction

The principles of the relational database systems are based on ideas from the mathematical theories about sets and logic. These principles were introduced by **Dr. E. F. Codd** in 1970 while he was working as a researcher at IBM. Dr Codd discovered that by applying the ideas from mathematics to database systems, these system could be studied and developed in a more rigorous fashion. Due to these principles the field of database design began to be studied as truly scientific field.

In 1985 Dr. E. F. Codd published an article that lists rules for how to determine whether a DBMS is relational and to what extent it is relational. These rules are now known as **Codd's 12 rules** (the rules are numbered starting from 0, so in fact there are a total of 13 rules). These rules provide a very useful yardstick for evaluating a relational system. One of these rules specifies that all relational database systems should have at least one database access language that can be used to manipulate the database and the data present in it. A set-oriented database language is what differentiates a Relational Database Management System (RDBMS) fro a Database Management System (DBMS). Most RDBMSs today implement SQL as their set-oriented database language. The term Structured Query Language (SQL) can be misleading for the beginners. The word Query suggests that the language is restricted to making inquiries or searches of the database. However SQL can be used to do much more. You can create tables, add data, delete data, splice data together, trigger actions based on changes to the database, and store your queries within your program or database. Database Administrators can use SQL to do the following:

- Modify a database's structure.
- Change system security settings
- Add user permissions on databases or tables
- Query a database for information
- Update the contents of a database.

There are two ways of interacting with a relational database, the first, which we have studied up to this point in the book, is to use a graphical interface called the Integrated Development Environment (IDE). The second is to do it using SQL. Every action you take in Access has an equivalent SQL statement that does the same thing. Studying SQL has two benefits first it lets you have a closer look at the relational structure and workings of the database that are hidden behind the GUI in Access, secondly it will help you write powerful queries which are not possible using the Access GUI.

10.1 The SQL Interface

Databases store and retrieve data using complex data structures and algorithms written in computer languages like C, and C++. SQL provides a simple way to the user to work with the database without knowing the internal workings of the database.

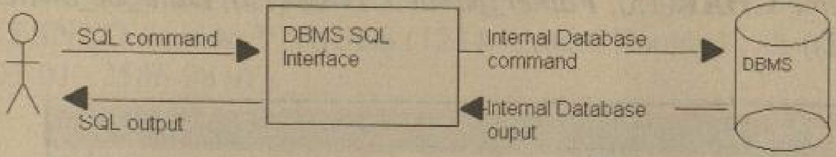
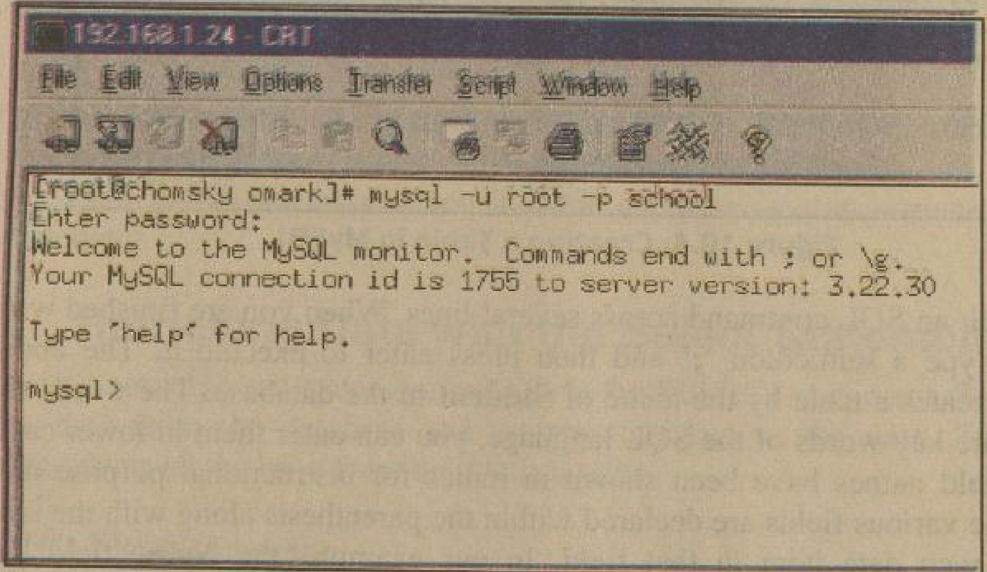

SQL command
SQL output
DBMS SQL Interface
Internal Database command
Internal Database ouput
DBMS

Figure 10.1: SQL Interface

As you can see from the diagram above the SQL interface takes the command from the user and converts it into the internal database command. The same SQL command will work on different database even though the internal database commands will vary from database to database. The following figure shows the SQL interface of a popular relational database management system called **MySQL**.


192.168.1.24 - CRT
File Edit View Options Transfer Script Window Help

```
[root@chomsky omark]# mysql -u root -p school
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1755 to server version: 3.22.30

Type 'help' for help.

mysql>
```

Figure 10.2: MySQL interface

SQL statements are entered at the prompt to interact with the database. We will now look at various SQL statements and create a table, similar to the one created in the chapter 3, using only SQL.

10.2 The CREATE Statement

The following figure represents the structure of the **Student** table in our **School database**:

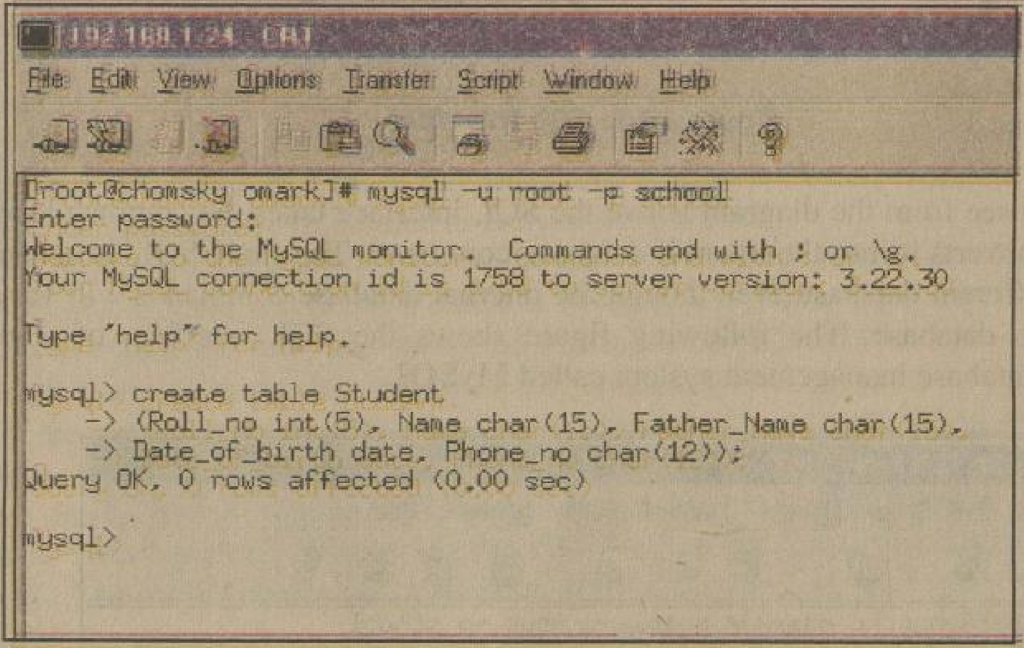
Roll_no	Name	Father_Name	Date_of_birth	Phone_no
1234	Abdul Hameed	Amir Hameed	1990-03-01	586-9830
1235	Usman Khan	Tariq Khan	1991-02-02	575-1312
1236	Irfan Chaudhry	Asim Chaudhry	1990-09-04	767-1247

1237	Abdul Hameed	Shahid Hameed	1990-04-02	555-5590

Figure 10.3: Sample Table

This table can be created in the database using the following SQL statements:

CREATE TABLE *Student*
(*Roll_no* **INT**(5), *Name* **CHAR**(15), *Father_Name* **CHAR**(15), *Date_of_birth* **DATE** , *Phone_no* **CHAR**(12))



```
[root@chomsky omark]# mysql -u root -p school
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1758 to server version: 3.22.30

Type 'help' for help.

mysql> create table Student
    -> (Roll_no int(5), Name char(15), Father_Name char(15),
    -> Date_of_birth date, Phone_no char(12));
Query OK, 0 rows affected (0.00 sec)

mysql>
```

Figure 10.4: Creating a Table in MySQL

You can split an SQL command across several lines. When you are finished writing the command, type a semicolon ';' and then press enter to execute it. The above SQL command creates a table by the name of **Student** in the database. The words shown in uppercase are key words of the SQL language, you can enter them in lower case if you like. The field names have been shown in italics for instructional purpose only. The names of the various fields are declared within the parenthesis along with the maximum length for each data item in that field. In our example, the **Name** field has been specified to consist of a maximum of 15 characters. Thus, any name with less than or equal to 15 letters of the alphabet can be entered in it. The word **CHAR** stands for *character* and it is a type of data defined in the database. Similarly, **INT** is another data type that stands for *integers* and is used in the above example to specify that roll numbers of students can only be entered as a pure number in the database, i.e. we can not have a letter of the alphabet as part of a roll number.

To create the same table with the **Roll_no** field set as the primary key use the following statement:

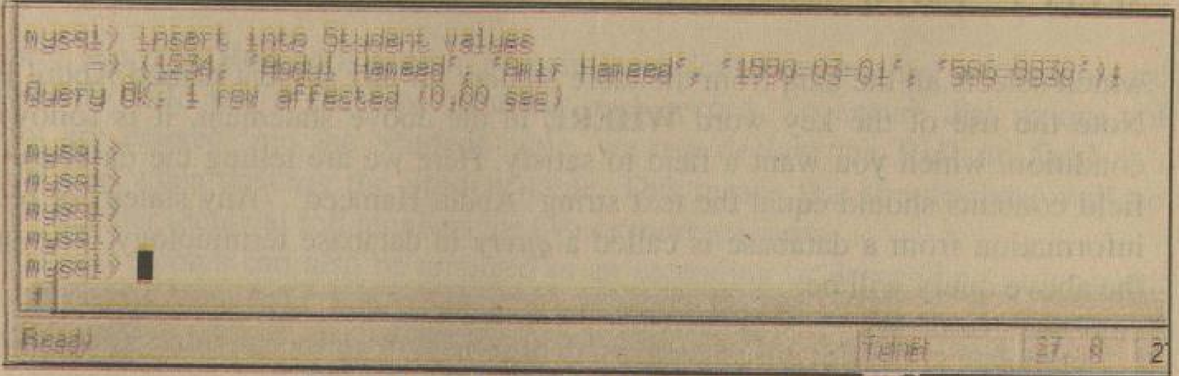
CREATE TABLE *Student*
(*Roll_no* **INT**(5) **PRIMARY KEY**, *Name* **CHAR**(15), *Father_Name* **CHAR**(15), *Date_of_birth* **DATE** , *Phone_no* **CHAR**(12))

Note the key word **PRIMARY KEY** with the **Roll_no** field.

10.3 Adding Data with the INSERT statement

Now that we have created a database it is time to put some data into it. This is done using the **insert** statement. For example to insert data about the student 'Abdul Hameed' we issue the following statement:

INSERT INTO *Student* **VALUES** (1234, 'Abdul Hameed', 'Amir Hameed', '1990-03-01', '586-9830')



```
mysql> insert into Student values
    -> (1234, 'Abdul Hameed', 'Amir Hameed', '1990-03-01', '586-9830');
Query OK, 1 row affected (0,00 sec)

mysql>
mysql>
mysql>
mysql>
mysql> █
```

Figure 10.5: Inserting Values into Table

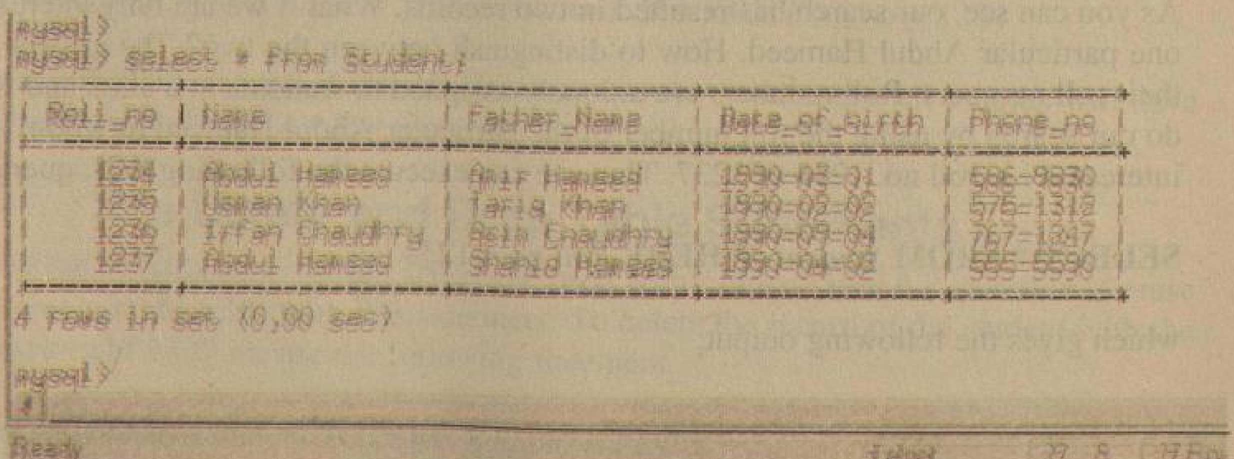
We add information about all the students in this fashion. Each insert statement adds a row to the table. This process is also called **populating** the database.

Once we have put all the information into the database we can retrieve it using the select statement

10.4 Retrieving Data with the Select Statement

The most commonly used statement in SQL is the SELECT statement. This statement retrieves data from the database and returns this data to the user. The following statement selects all the rows from the Student table.

SELECT * FROM *Student*



```
mysql>
mysql> select * from Student;
+---------+-----------------+----------------+---------------+----------+
| Roll_no | Name            | Father_Name    | Date_of_birth | Phone_no |
+---------+-----------------+----------------+---------------+----------+
|    1234 | Abdul Hameed    | Amir Hameed    | 1990-03-01    | 586-9830 |
|    1235 | Usman Khan      | Tariq Khan     | 1990-02-02    | 575-1312 |
|    1236 | Irfan Chaudhry  | Asim Chaudhry  | 1990-09-04    | 767-1347 |
|    1237 | Abdul Hameed    | Shahid Hameed  | 1990-04-02    | 555-5590 |
+---------+-----------------+----------------+---------------+----------+
4 rows in set (0,00 sec)

mysql>
```

Figure 10.6: Retrieving Information

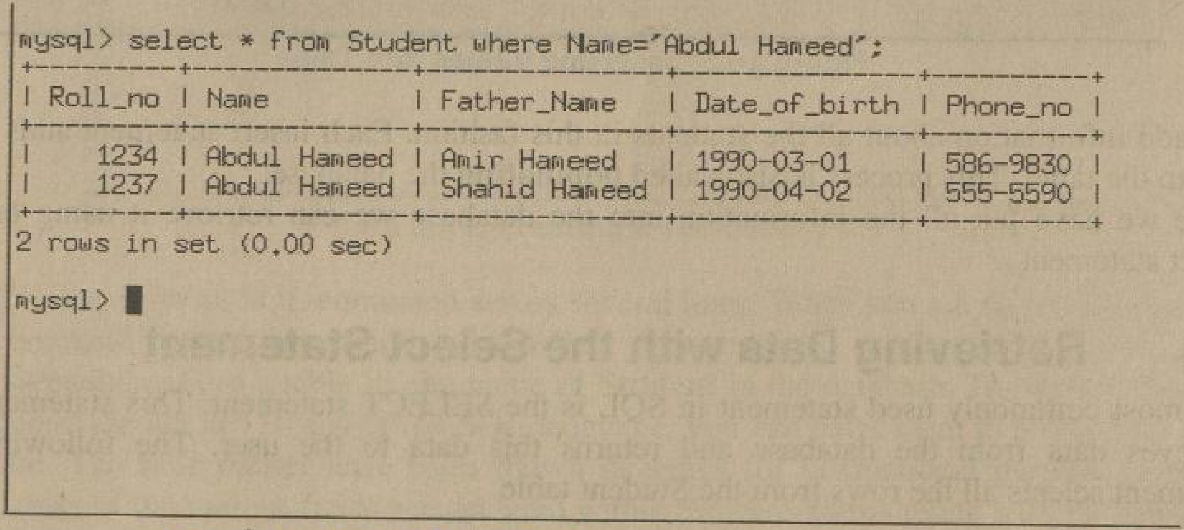
The * in the above statement means that all columns should be selected. To select specific columns just type their names instead of the *. For example if you wanted to retrieve only the Roll_no and Name fields from all the records the statement would be

SELECT *Roll_no*, *Name* **FROM** *Student*

Now it is quite likely that there are two or more students in a class who have the same name, say Abdul Hameed, as in the above example. Suppose we want to get all the information about all of the Abdul Hameed's. We could write the following statement:

SELECT * FROM *Student* **WHERE** *Name* = 'Abdul Hameed'

which selects all the data from the table *Student* for any student named Abdul Hameed. Note the use of the key word **WHERE** in the above statement, it is followed by a condition, which you want a field to satisfy. Here we are telling the database that the field contents should equal the text string 'Abdul Hameed'. Any statement requesting information from a database is called a ***query*** in database terminology. The output of the above query will be:



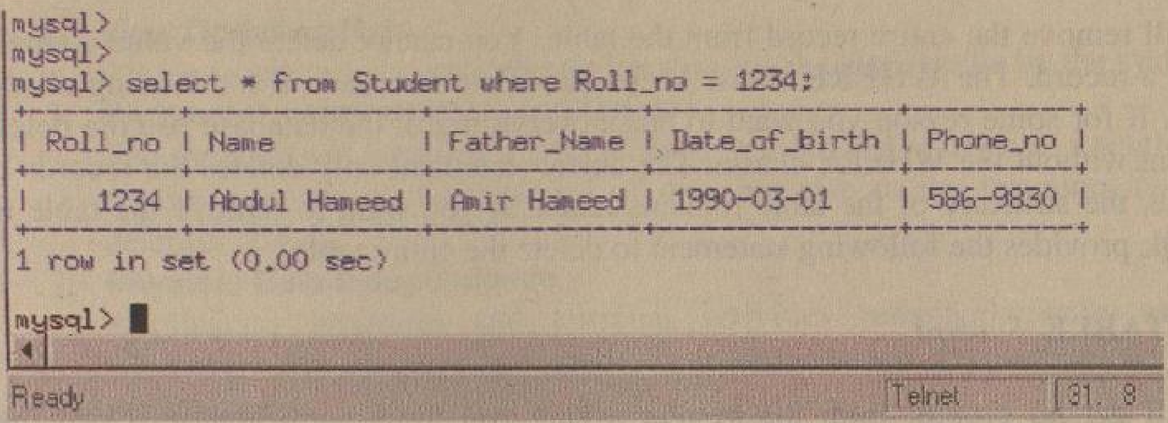
```
mysql> select * from Student where Name='Abdul Hameed';
+---------+--------------+---------------+---------------+----------+
| Roll_no | Name         | Father_Name   | Date_of_birth | Phone_no |
+---------+--------------+---------------+---------------+----------+
|    1234 | Abdul Hameed | Amir Hameed   | 1990-03-01    | 586-9830 |
|    1237 | Abdul Hameed | Shahid Hameed | 1990-04-02    | 555-5590 |
+---------+--------------+---------------+---------------+----------+
2 rows in set (0.00 sec)

mysql> █
```

Figure 10.7: Retrieving Information according to some criteria

As you can see, our search has resulted in two records. What if we are only interested in one particular Abdul Hameed. How to distinguish between the two? The answer is: by their roll numbers. Roll numbers are uniquely assigned to students in a class and we can do our search by using the roll number of the particular Abdul Hameed in whom we are interested i.e. roll no. 1234 or 1237. Thus we can execute the following SQL query:

SELECT * FROM *Student* **WHERE** *Roll no* = 1234

which gives the following output:



```
mysql>
mysql>
mysql> select * from Student where Roll_no = 1234;
+---------+--------------+-------------+-------------------+----------+
| Roll_no | Name         | Father_Name | Date_of_birth     | Phone_no |
+---------+--------------+-------------+-------------------+----------+
|    1234 | Abdul Hameed | Amir Hameed | 1990-03-01        | 586-9830 |
+---------+--------------+-------------+-------------------+----------+
1 row in set (0.00 sec)

mysql> █
```

Figure 10.8: Output of the Select Query

Over here, the field **Roll no** is used to pinpoint a particular record, and just one record. The **Roll no** field has a different value for each record. To ensure that unique roll numbers are entered into the **Student** table, we can declare the **Roll no** field as a **primary key** when creating the **Student** table. This means that the database will not allow us to enter identical roll numbers for two or more students.

The retrieved records can also be arranged in an ascending or descending order based on the values in some field. For example to see the records of all the students arranged in an ascending order according to their birth dates execute the following statement

SELECT * FROM *Student* **ORDER BY** *Date_of_birth* **ASC**

The key words **ORDER BY** followed by a field name tell the database to retrieve the records in an order based on the given field and the key word **ASC** tells the database that the order should be ascending if you wanted to retrieve the records in a descending order use the word **DSC**

10.5 Modifying Data with the UPDATE Statement

The **UPDATE** statement is used to change the values of fields in the existing records. For example to change the name of the student whose roll no is 1234 we use the following statement

UPDATE *Student* **SET** *name* = 'Omar Hameed' **WHERE** *Roll_no* = 1234

Note that if don't use the **WHERE** clause in the above statement all records will have their name field set to the value *Omar Hameed.*

10.6 The Delete and Drop Table Statements

The **DELETE** statement should always be used with caution since it is possible to erase all the rows in the table with this statement. To delete the record of the student with the roll number of 1237 we use the following statement

DELETE FROM *Student* **WHERE** *Roll_no* = 1237

This will remove the entire record from the table. You cannot delete the value a single field of a record. The **WHERE** clause keeps the delete statement from deleting all the records. If for some reason you want to erase everything in the database use the delete statement without the WHERE clause. The delete statement only deletes the records in the table, the structure of the table remains intact. If you want to remove the table as well SQL provides the following statement to delete the entire table.

DROP TABLE *Student*

You will get no prompt from the system asking you for a confirmation. Once this command has been issued, the table is permanently deleted.

10.7 CODD's 12 RULES

Dr. E. F. Codd, the originator of the relational data model, published a two-part article in ComputerWorld (Codd, 1985) that lists 12 rules for how to determine whether a DBMS is relational and to what extent it is relational. These rules provide a very useful yardstick for evaluating a relational system. Codd also mentions that, according to these rules, no fully relational system is available yet. In particular, rules 6, 9, 10, 11, and 12 are difficult to satisfy.

10.7.1 THE 12 RULES

1. **The Information Rule**
   All information in a relational database is represented explicitly at the logical level in exactly one way—by values in tables.
2. **Guaranteed Access Rule**
   Each and every datum (atomic value) in a relational database is guaranteed to be logically accessible by resorting to a table name, primary key value, and column name.
3. **Systematic Treatment of Null Values**
   Null values (distinct from empty character string or a string of blank characters and distinct from zero or any other number) are supported in the fully relational DBMS for representing missing information in a systematic way, independent of data type.
4. **Dynamic On-line Catalog Based on the Relational Model**
   The database description is represented at the logical level in the same way as ordinary data, so authorized users can apply the same relational language to its interrogation as they apply to regular data.
5. **Comprehensive Data Sub-language Rule**
   A relational system may support several languages and various modes of terminal use (for example, the fill-in-blanks mode). However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and whose ability to support all of the following is comprehensible: data definition, view definition, data manipulation (interactive and by program), integrity constraints, and transaction boundaries (begin, commit, and rollback).

6. **View Updating Rule**
   All views that are theoretically updateable are also updateable by the system.
7. **High-level Insert, Update, and Delete**
   The capability of handling a base relation or a derived relation as a single operand applies nor only to the retrieval of data but also to the insertion, update, and deletion of data.
8. **Physical Data Independence**
   Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representation or access methods.
9. **Logical Data Independence**
   Application programs and terminal activities remain logically unimpaired when information preserving changes of any kind that theoretically permit unimpairment are made to the base tables.
10. **Integrity Independence**
    Integrity constraints specific to a particular relational database must be definable in the relational data sub-language and storable in the catalog, not in the application programs. A minimum of the following two integrity constraints must be supported:
    1. **Entity integrity**: No component of a primary key is allowed to have a null value.
    2. Referential integrity: For each distinct nonnull foreign key value in a relational database, there must exist a matching primary key value from the same domain.
11. **Distribution Independence**
    A relational DBMS has distribution independence. Distribution independence implies that users should not have to be aware of whether a database is distributed.
12. **Nonsubversion Rule**
    If a relational system has a low-level (single-record-at-a-time) language, that low-level language cannot be used to subvert or bypass the integrity rules or constraints expressed in the higher-level (multiple-records-at-a-time) relational language.

Note: There is a rider to these 12 rules known as **Rule Zero**: "For any system that is claimed to be a relational database management system, that system must be able to manage data entirely through its relational capabilities."
On the basis of the above rules, there is no fully relational DBMS available today.

Chapter 11

Introduction to Computer Science

Overview

The objectives of this course are to help you learn simple computer programming methods and to show how to apply very basic computer science principles to solve problems. We will use the C programming language for both these objectives. Computer languages like C are designed to shield the programmer from much of the hardware details of a computer. However a basic knowledge of how computers work will help you better understand the rules of the language and the various programming principles. We therefore start this chapter with a study of various components of a computer and how they are organized. After we have discussed the physical architecture of the computer we will study the concept of computer languages.

11.1 Computer organization

Modern computers are similar in architecture to a computing machine designed by von Neumann in 1946. Therefore they are said to have a von Neumann architecture. The von Neumann model for a computer is shown in the Figure 11.1.

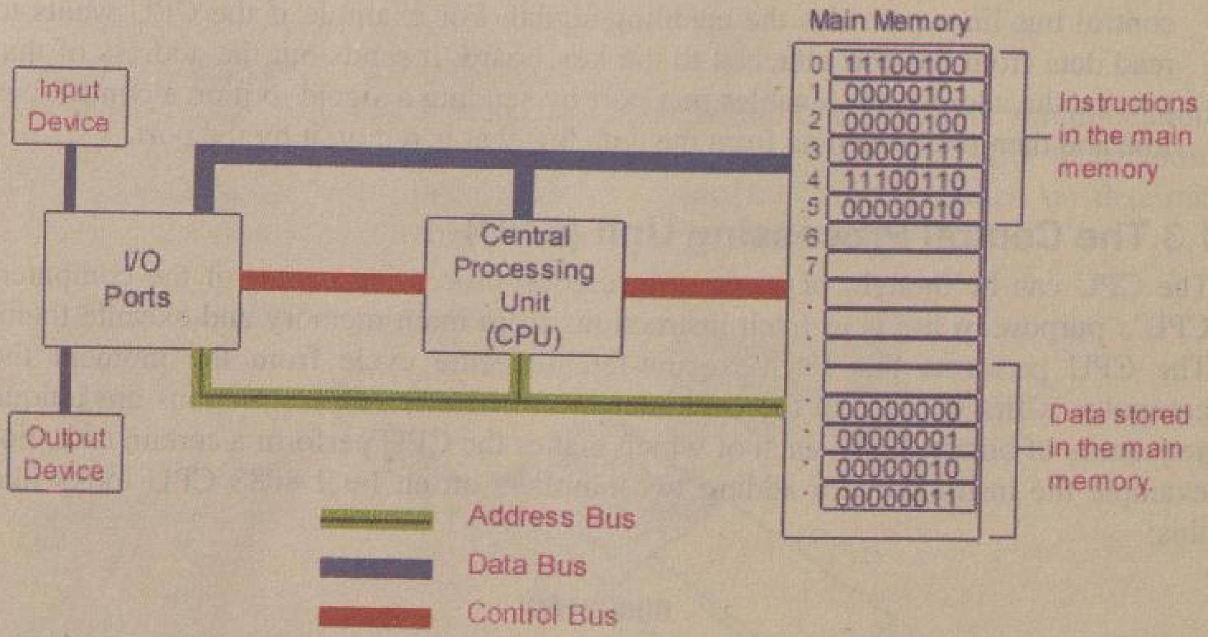

Main Memory
0 11100100
1 00000101
2 00000100
3 00000111
4 11100110
5 00000010
6
7
Instructions in the main memory
Input Device
I/O Ports
Central Processing Unit (CPU)
Output Device
00000000
00000001
00000010
00000011
Data stored in the main memory.
Address Bus
Data Bus
Control Bus

Figure 11.1: The von Neumann Computer model

Various components of this computer model are described in detail below:

11.1.2 System Buses

System buses are electrical paths used to carry various electrical signals between the components of a computer. These buses can be categorized according to the information they carry.

- **Address Bus**
  The address bus consists of parallel signal lines (you can think of them as wires) connected to the central processing unit (CPU), the main memory and the I/O ports. On these lines the CPU sends out the address of the memory location or I/O port that is to be written to or read from. The address is sent out in binary form. Thus each address line has either a '1' or a '0' value put on to it. The number of address line present determines the number of memory locations or I/O ports that the CPU can address. If the CPU has N address lines then it can address 2^N memory and I/O locations.
- **Data Bus**
  The data bus like the address bus consists of parallel signal lines connected to the CPU, the main memory and the I/O ports. The data bus lines are bi-directional. This means that the CPU can read data in on these lines from memory or from a port as well as send data out on these lines to a memory location or to a port. Many devices in a system, keyboard, memory, mouse etc will have their outputs connected to the data bus, but the outputs of only one device at a time will be enabled.
- **Control Bus**
  As was mentioned in the section on data bus, many devices are connected to the data bus but only one of these is enabled at a give time. The CPU through the control bus lines provides the enabling signal. For example if the CPU wants to read data from the port attached to the key board, it sends out the address of that port on the address bus, enables that port by sending a signal to it on a control bus line and then reads the data from the data bus that is put on it by the port.

11.1.3 The Central Processing Unit (CPU)

The CPU can be thought of as the information-processing center of the computer. CPU's purpose in life is to fetch instructions from main memory and execute them. The CPU performs this fetch-execute-fetch-execute cycle from the moment the computer is first turned till the computer is turned off. The instructions are unique sequences of binary digits, each of which makes the CPU perform a certain task. For example the instruction for adding two numbers on an Intel 8088 CPU looks like this:

00000100

Figure 11.2: Intel 8088 microprocessor instruction for adding two numbers

Whenever an Intel 8088 CPU reads this instruction it is designed to perform the task of adding two numbers. The designers of the 8088 CPU have designed it to perform hundreds of such tasks e.g. addition, subtraction, multiplication, data storage etc. Each task has its own unique binary instruction like the one in the Figure 11.2. The set of all these binary instructions is called the ***Instruction Set***. A set of related instructions that solve a particular problem is called a ***program***. The instructions are stored in the main memory in the form of binary numbers. To better understand the working of the CPU we can think of it as divided into the following three parts.

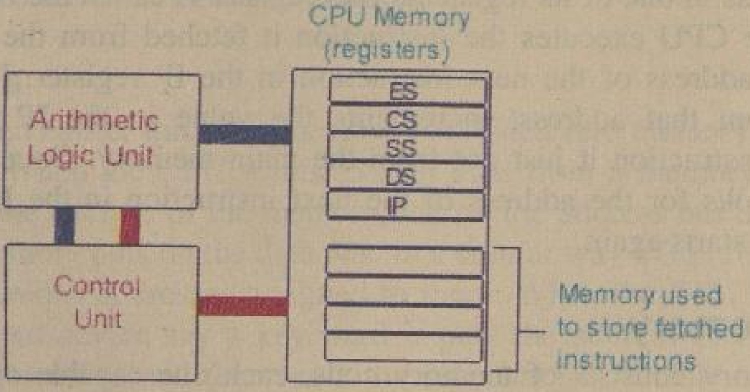

CPU Memory
(registers)
Arithmetic
Logic Unit
ES
CS
SS
DS
IP
Control
Unit
Memory used
to store fetched
instructions

Figure 11.3: Simplified model of the CPU

- **CPU Memory (Registers)**
  The CPU has memory of its own. This memory is divided into small parts. Each part is called a register. If the CPU wants to store something, say a number or a main memory address it stores it in one of its registers.
- **Control Unit**
  The control unit acts like a supervisor in the CPU. It is the job of the control unit to fetch instructions from the main memory, interpret the instructions and then direct the execution of the instruction.
- **Arithmetic-Logic Unit (ALU)**
  When arithmetic calculations, such as addition, subtraction, multiplication, or division, must be performed, the control unit directs the ALU to perform the tasks. Logical operations, such as comparing tow quantities or determining whether a condition is true or false, are also performed by the ALU.

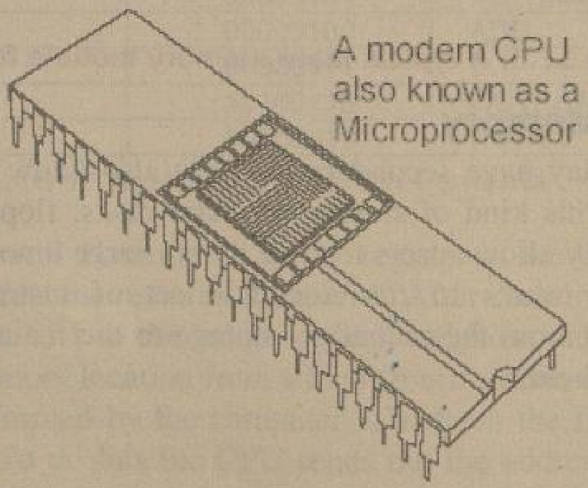

A modern CPU
also known as a
Microprocessor

Figure 11.4: Example of a CPU found inside modern PCs

Let us now look at the fetch-execute cycle in a little more detail. When you start a computer, the first thing the CPU does is to fetch an instruction from a default main memory address. Every time you start a computer the same thing happens, the CPU looks for the first instruction in the same default main memory address. The CPU then increments this address to point to the next instruction and stores the resultant memory address in one of its registers. This register is called the ***instruction pointer (IP)***. After the CPU executes the instruction it fetched from the main memory, it looks for the address of the next instruction in the IP register. It fetches the next instruction from that address, increments the value in the IP register and then executes the instruction it just got from the main memory. Once the instruction is complete it looks for the address of the next instruction in the IP register and the whole process starts again.

11.1.4 Main Memory

The main memory consists of memory cells, each one capable of storing a binary number. Each memory cell is assigned a unique number known as a memory address. If the microprocessor wants to read the binary number in a memory cell it puts the address of the cell on the address bus, the main memory responds by putting the contents of the memory cell, with that address, onto the data bus. The CPU then reads the number that was stored in the memory cell from the data bus. In personal computers the main memory is called the ***random access memory (RAM)***. Instructions that are to be run are first loaded into the RAM and are then executed by the CPU. The Figure 11.5 shows a typical RAM found in personal computers.

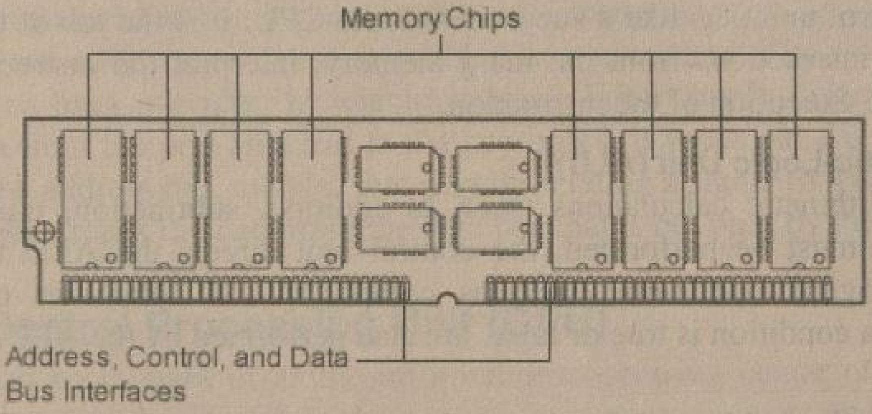


Figure 11.5: A typical main memory module for PCs

11.1.5 Secondary Memory

Most modern PCs today have secondary memory also know as ***secondary storage***. Typical examples of this kind of storage are hard disks, floppy disks and cd-roms. This additional memory allows users to store a very large amount of information and data. When the user wants to run a certain set of instructions (program) the instructions are copied from the secondary storage to the main memory and then the CPU starts executing them.

11.1.6 Input/Output Devices

The input/output (I/O) devices allow communication between the user and the computer. Examples of input devices are keyboards and mice. Examples of output devices are video displays and printers. I/O devices provide us with the means to enter data for a computation and to observe the results of that computation. Input and output devices connect to the CPU through I/O ports.

11.1.7 I/O Ports

As was mentioned earlier the main memory is divided into memory cells each with a unique address. When the CPU wants to read data from a memory cell in the main memory it puts the address of the memory cell on the address bus and then reads the data the main memory puts on the data bus. In a similar way I/O ports connected to the computer have unique addresses assigned to them. When the CPU wants to read the data from an input device say a keyboard it puts the address of the Input port the keyboard in connected to and then reads the data that is available on the data bus. If you look at the backside of a personal computer you will be able to see the connections for various I/O ports. Some of these are shown below.

11.2 Instruction Execution Cycle

Now that we have an idea of all the major components of a computer let us see how all of them work together to run (*execute*) a simple program (*a set of instructions*). The examples are for a simple Intel 8088-based computer. The instructions will do the following:

1. Input a number from a keyboard connected to a port at address 5.
2. Add 7 to the value read from the port.
3. Output the result to a display connected to a port at address 2.

The following Figure 11.6 shows the contents of the memory location where the instructions are stored.

Memory Address	Memory Contents	Operation
0	11100100	INPUT FROM
1	00000101	PORT 5
2	00000100	ADD
3	00000111	7
4	11100110	OUTPUT TO
5	00000010	PORT 2

Figure 11.6: Memory Location Contents

For this example we will assume that the IP register in the CPU is initialized to the value 0 which represents the memory location 0. After each instruction/data is fetched the CPU adds a value of 1 to the value in the IP register, thus the IP register always points to the next memory location from which an instruction or data is to be fetched.

The first action performed by the computer is to fetch the first instruction from the memory location 0. To do this the CPU sends out the address 0 on the address bus. The CPU then sends out a control signal on the control bus to the memory. This causes the memory to output the first instructions '11100100' on the data bus. The CPU reads in the instruction from the data bus and ***decodes*** it. By decode we mean that the control unit in the CPU determines, from the instruction just read in, what actions it is supposed to take. In this case the CPU determines that the instruction is asking it to input data from a port. It also determines that it needs the address of the port it is supposed to read the data from. The CPU must fetch from memory the input port address. To do this, the CPU sends out the address of the next memory location (remember that the CPU gets this value from the IP register which now has the value of 1 in it, having been incremented after the last fetch) on the address bus and sends a control signal to the main memory again. This enables the memory to put the value of 5 stored at the address 1 onto the data bus. The CPU reads in the value 5 and now has all the information to get the input data from the port. To execute the input instruction the CPU sends out the port address 5 on the address bus and sends a control signal to the port on the control bus. The addressed port device then puts some binary data on the data bus, which is read by the CPU. The CPU stores this data in one of its internal memory locations or registers. This internal register is known as the ***accumulator***.

After completing this instruction the CPU gets the next instruction from the memory address 2. When the CPU decodes this instruction it finds out that it is supposed to add some number to the number stored in the accumulator register. There is already a value in the accumulator register to get the second number the CPU must read the data from the next memory location. The CPU performs another fetch cycle and gets the value of 7 from the memory location 3. Now the CPU has both the numbers it has been instructed to add. The control unit in the CPU gives the two numbers to the ALU, which adds the two numbers and stores the result back into the accumulator register.

The next instruction that the CPU fetches from the address 4 tells it to output the value in the accumulator register onto a port. The CPU decodes the instruction and finds that it needs a port address. It gets this address (value 2) by fetching the next binary number from the main memory cell with an address of 5. Once it has the address it puts the value in the accumulator register onto the data bus and sends a control signal to the device attached on port no 2 to enable it to read the data. The output device reads the data and starts displaying the result of the addition.

All of the step described above may seem like a great amount of work just to add two numbers, however most modern computers can perform all of these in less than a microsecond.

11.3 Programming Languages

Now that you have an idea of how computers perform in response to various instructions fed to them, it is time to learn how to write programs to solve various problems using the computer. There are three ways to program a computer.

11.3.1 Machine Language

As we saw in the last section, when we added two numbers and displayed their result, instructions in the main memory are stored as a sequence of binary numbers. The program at this level is expressed in machine language. To write a program using machine language we would first have to memorize the instruction set of the CPU we are working on (note that different CPUs have different instruction sets). This is a difficult if not an impossible job for a programmer. What makes the job even more difficult is that the machine language only provides instructions for very simple operations like adding two numbers present in the CPU's registers or moving a number from a memory location to a register. Even a simple addition requires four or five instructions. Writing a complex program is therefore almost impossible in machine language.

11.3.2 Assembly Language

To make the task of programming easier assembly language was created. Assembly language uses letters to represent each instruction type. For example the add instruction is represented by ADD and the instruction for moving data from a memory location to a CPU register is represented by MOV. In Figure 11.6 we gave the machine code for adding the value 7 to the value in the accumulator register. Here is the assembly language code for that

```
ADD        AL, 07
```

As you can see this is much easier to remember that a bunch of binary numbers. Programs written in assembly language can easily be converted to machine code using programs called assemblers. This machine code is then used to run the program. Assembly language is an improvement on machine language but it still does not allow us to write large programs efficiently and easily. When writing programs we want to concentrate on the method of solving the problem at hand not on the values in the registers and memory locations of the machine. Another disadvantage of writing programs in assembly language is that programs written for one CPU will not work on a different CPU since their instruction sets don't match. In other words assembly language programs are not portable. To solve these problems we use higher-level languages.

11.3.3 Higher-Level Languages

Higher-level languages allow us to think about problems in a more natural and familiar way. Let us look at the instructions in a higher-level language for reading a value from the keyboard adding 7 to it and displaying the result on the monitor.

```
read (inupt);
result : = input + 7;
write (result);
```

Anyone looking at these instructions can determine what the instructions are meant to do because of the readable familiar notations used. The computer however needs the instructions in machine language form. To convert a program written in a higher-level language to an equivalent program in machine language we use a compiler. For example C language compilers are available for computers ranging from personal computers to supercomputers. Suppose we write a program, in the C language, on a personal computer that has an Intel 8088 CPU. We then translate this program to the machine language of the Intel 8088 CPU by compiling it with the C language compiler available on the personal computer. Now If we want to run the same program on a supercomputer, we just have to copy the C language program new machine and compile it using the C language compiler available on it. This will give us the same program in the machine language of the supercomputer. It is important to realize that we did not have to rewrite the program. We just used the compilers on the two machines to do the job for us. In other words programs written in the C language are portable. Note however that some one has to write a C language compiler for each computer that has a different CPU in it, but this has to be done only once. Other examples of higher-level languages are C++, Java, Lisp, Perl, Fortran and Pascal.

11.4 Characteristics of Higher-Level Languages

Virtually all programming today is done in higher-level languages. Higher-level programming languages have the following characteristics, which help the programmers achieve their goal more efficiently:

- **Readable, natural language like notation**
  As already mentioned, higher-level languages are much closer to the languages humans use in their design. Instead of getting bogged down in the intricacies of the machine language the programmer can concentrate on the problem to be solved. The readability and ease of programming has also resulted in a tremendous increase in the number of computer programmers. This has made possible writing of large and complex software packages, such as operating systems on which literally hundreds of programmers work at the same time.
- **Portability (Machine independence)**
  We have seen how a C language program written on one machine can be easily run on another machine that has a different architecture. This portability has helped in the development of new programming techniques and principles. A program written in America can immediately be transferred using the Internet and run on a computer in say Australia. Computer scientists can therefore exchange ideas; this helps them to develop better and more efficient programs.
- **Availability of program libraries**
  Due to the readability and portability of programs written in higher-level languages you can use code written by other programmers. For example, suppose you wanted to calculate the square root of a number as part of your program, there is no need to write your own code to do this. Programmers have already developed code for thousands of mathematical functions. You can just plug the available code for the square root function in your program. This library of code available to a programmer reduces the development time of a program drastically.
- **Relatively easy error detection**
  The readability of the programs helps to find errors. Moreover we can design programs so that their correctness can be understood in terms of their structure. Since we are free to think about the structure of the program without thinking about the architecture of the machine, we can organize the computations in our program in such a way that our limited powers are sufficient to guarantee that the program will achieve its desired objective.

Exercises

1. List the major components of a von Neumann computer model.
2. What Steps does the processor go through to execute a statement?
3. Where is the accumulator located and what is its purpose?
4. Programming languages can be divided into how many kinds? Give an example of each one of them.
5. What are the advantages and disadvantages of using higher-level languages?

Chapter 12

Basic Structure of a Program

12.1 Our First C Program

The C language is classified as an ***imperative*** language. Imperative languages are action oriented; that is, a computation is viewed as a sequence of actions. To better understand the structure of a C language program and its execution, we consider here a simple program that prints the phrase "My first C program" on the screen. The complete program is given below:

```
#include <stdio.h>

int main(void)
{
    printf("My first C program\n");
    return 0;
}
```

Figure 12.1

Using a text editor we write this code into a simple text file and the save it as `first.c`. This becomes our program source file. When the source file is compiled, an executable file is produced. On DOS/Windows operating system the executable file will be called `first.exe`. When we run the program by typing its name on the command prompt, it prints on the screen the following sentence.

```
My first C program
```

Analysis of the Program

We will now study the program line by line to make the working of the program clear.

- `#include <stdio.h>`

  To convert a C language source file into an executable machine language file we use a compiler. The first action of the compiler is to look in the program file for lines beginning with a # and process these lines. The part of the compiler that does this is known as the ***preprocessor***. When the preprocessor encounters a line starting with `#include` it looks for a file whose name appears after the `#include` directive between the angle brackets (< >). In this case the file name is `stdio.h`. The preprocessor then replaces the `#include <stdio.h>` line in the program file with the contents of the `stdio.h` file. You can think of it as a simple copy and paste procedure from the `stdio.h` file into our program file. `stdio` stands for *standard input output*. Remember that our program prints a sentence on our output device, the screen. The `stdio.h` file contains information about the output function `printf()` that we have used to print our sentence on the screen. Files, such as `stdio.h`, that have the extension '.h' are known as ***header files***. It is a convention to use the '.h' extension although it is not necessary. These files are provided by the C compiler system and the compiler knows where to look for them. It is important to realise that the line beginning with a # sign are directives or instructions to the preprocessor in the compiler; they are not part of the C language. The header files although do contain C language directives.

- `int main(void)`

  All C programs contain `main()`. It indicates the point at which program execution starts. As we know programs are run from within an operating system. For example we run programs in the DOS operating system from the familiar **C:\>** prompt. When a program written in C language finishes running it returns a value to the operating system. This happens in the background so we don't see it happening. The operating system uses this value to determine whether the program ran successfully or not. The keyword `int` before `main()` stands for "integer". It is our way of telling the compiler that our program will return an integer value to the operating system when it stops execution.

  When we want to delete a file from the current directory in DOS we use the command C:\>delete *filename* or in other words we give the *delete* program the name of the file to delete. In computer science we say that the delete program takes the name of a file as an ***argument***. The *dir* command takes no arguments, you just type dir on the command prompt and the command executes. The key word `void` in `main(void)` is our way of telling the compiler that our program takes no arguments, we just type its name on the command line press enter and it runs.

- `{`

  Braces are used to group all the instructions in our C program together. The left brace `{` after `main()` can be thought of as the place where the program instructions start.

- `printf("My first C program\n");`

  The above instruction has two parts, `printf()` and the sentence `"My first C program\n"`. `printf()` is an example of a function that is provided by the C language system. For the moment you can think of `printf()` as a command that is available in the C language. The command `printf()` prints whatever sentence you give it between the parentheses. We included the `stdio.h` file to provide the compiler with certain information about `printf()`.

  The sentence `"My first C program\n"` is an example of what is know as a ***string constant***. A string constant in C is a series of characters surrounded by double quotes. This sentence or string is called an *argument* to the `printf()` function and it controls what gets printed. The two characters `\n` at the end of the string represent a ***newline***. It tells the `printf()` function to move the cursor on

the screen to the beginning of the next line. The semicolon '`;`' at the end of the line tells the compiler that the instruction is now complete. In C an instruction followed by a semicolon is known as a ***statement***.
When this whole line is executed the `printf()` command or function prints the sentence **`My first C program`** on the screen and then moves the cursor onto the beginning of the next line. Note that the double quotes "" don't get printed. They are there just to tell the `printf()` function that the argument is a string constant.

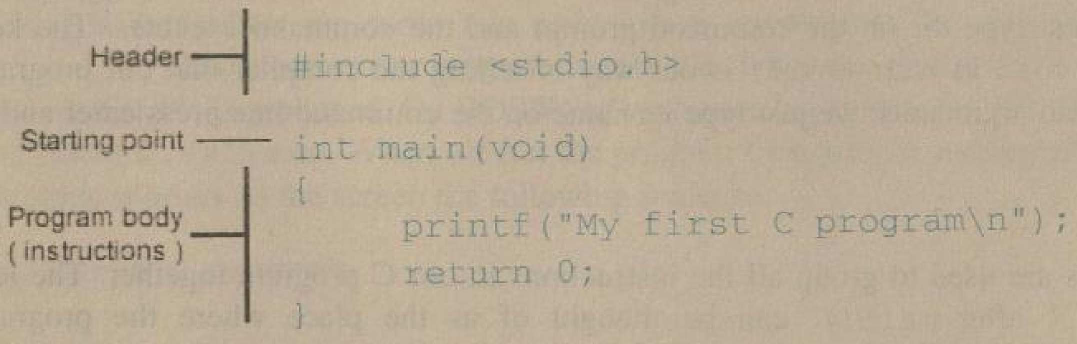
- `return 0;`

  This is the return statement (the instruction `return 0` followed by a semicolon is a statement). Remember the keyword `int` before the `main()`? We told you that when the program ends execution it returns a value to the operating system. The above statement causes the value zero to be returned to the operating system.

- `}`

  The right brace matches the left brace which we put in after `main()` it tells the compiler that the group of instructions in our program has ended.

The following diagram represents the basic structure of a C program



```
#include <stdio.h>

int main(void)
{
        printf("My first C program\n");
        return 0;
}
```

Figure 12.2: C program Structure

Exercises

1. Find out the errors produced when you leave out the `#include <stdio.h>` line from the program given in Figure 12.1. What if you leave out the semicolon at the end of the `printf()` statement?

2. Find out where the header files in your system are located?

3. What do the lines beginning with the symbol '#' used for? Which part of the compiler handles these lines?

4. Find out the output of the following program

```
#include <stdio.h>
int main (void)
{
    printf("My first");
    printf(" C ");
    printf("program\n");
    return 0;
}
```

Compare the output of this program with the program in Figure 12.1. What does it tell you about the printf() function?

5. Instead of writing the `printf()` on a single line, break it into two lines by including a line break (press the **Enter** key) in the middle of the argument string. What happens when you compile the program?

```
#include <stdio.h>
int main(void)
{
printf("My first
C program\n");
return 0;
}
```

6. A backslash at the end of any line is supposed to have the effect of continuing it to the next line. See if this works on your system. Compile and run the following program.

```
#include <stdio.h>
int main(void)
{
printf("My first\
C program\n");
return 0;
}
```

7. During the execution of a program it may be required to stop the program from continuing. For example it is possible to make a mistake in programming and have the computer stuck in a cycle performing the same instructions over and over again. In such situations we say that the program is in an ***infinite loop***. To interrupt such programs usually the control-c combination works on most systems. The following program contains an infinite loop. Run it and see if the control-c (press the keyboard keys marked **Ctrl** and **c** at the same time) interrupt gets you out of it.

```
#include <stdio.h>
int main(void)
{
    while (1)
    {
     printf("This program is in an infinite loop\n");
    };
    return 0;
}
```

Chapter 13

Lexical Element of C

13.1 The Language

In the preceding chapter we looked at a very simple program written in C language that just printed a message on our screen. Indeed the C language can do much more than that. In fact it is one of the most powerful computer programming language available today. It also is a language that is quite easy to learn, if not master. A powerful computer language however, must allow many more facilities to allow us to create useful programs. Learning a computer language is quite similar to learning any other language; i.e. We first learn its alphabet, grammar and its vocabulary. In fact the contents of a programming language are much smaller than a normal language.

In this chapter we will learn fundamental components of the C language. Let us however, first look at another program written in C that does a little more than printing a message on the screen.

```
/* Converting miles into kilometers */

#include <stdio.h>

int main(void)
{
    int miles;
    float kilometres;

    miles = 300;
    kilometres = 1.609 * miles;
     printf("\n300 miles equals %f
kilometres.\n\n",kilometres);
    return 0;
}
```

Fig. 13.1 Another Simple C Program.

The output of the program is ⇒ **300 miles equals 482.7 kilometres.**

Analysis of the Program

❑ `/* Converting miles into kilometres */`

Anything written between the characters /* and */ is a comment and is ignored by the compiler. You can write whatever information you like as a comment it will have no effect on the working of the program.

- `#include <stdio.h>`

  You have already been introduced to this directive in the previous example in chapter 2.

- `int miles;`

  `Whenever` we need to reserve memory in a computer where we can save numbers or other types of data we must declare it first so that the compiler can reserve appropriate amount of space. In order to do so we must inform the compiler (i) what type of data will be stored in the location/memory being reserved, and (ii) by what name will this memory space be called in our program. Such a reserved space inside a program is called a 'variable' simply because after having reserved the space we may use it in any manner we wish and change it any number of times in our program. Thus in order to use variables in a C program you first have to tell the compiler the name of the variable and its type. This is done through a declaration statement. In our program `int miles;` is a declaration statement that informs the compiler that we are going to use a variable named `miles` in our program. The keyword "`int`" tells the compiler that the variable `miles` will store integer values only i.e. whole numbers without any decimal or fractional part.

- `float kilometres;`

  This is another alternative variable declaration statement. `float` is a keyword and it informs the compiler that the variable kilometres will store real values. Real values are numbers that have a decimal point in them e.g. 3.14 and 40.0.

- `miles = 300;`

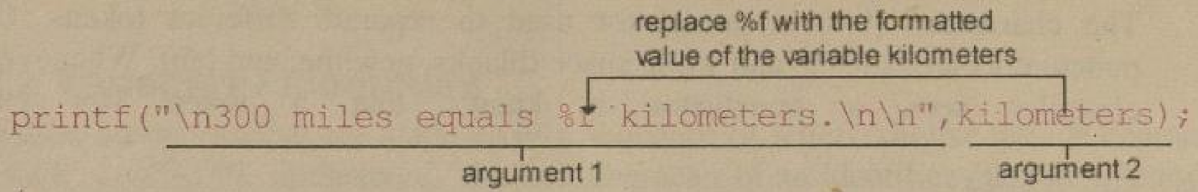
  This is an assignment statement. The equal sign '=' is an assignment operator. The number 300 is an integer constant. The statement causes the value 300 to be stored in the variable `miles` or in other words that the value 300 is assigned to the variable `miles`.

- `kilometres = 1.609 * miles;`

  This is another assignment statement. The result of the mathematical expression **`1.609 * miles`** on the right side of the equal sign is assigned to the variable `kilometres`. The operator * stands for multiplication. The number 1.609 is multiplied to the value of the variable `miles` (300) and the result (482.7) is assigned to the variable `kilometer`.

- `printf("\n300 miles equals %f kilometres.\n\n",kilometres);`

  This statement is similar to the one that we studied in the last chapter. It prints a sentence on the screen. This print statement takes two arguments as shown below


replace %f with the formatted
value of the variable kilometers
argument 1
argument 2

```
printf("\n300 miles equals %f kilometers.\n\n",kilometers);
{
```

The `%f` in the first argument is called the ***format***. The formats in the first argument, if any, are matched with the remaining arguments to the `printf()` function. In this case, `%f` is matched with the argument `kilometres`. It means that the value of the variable `kilometres` is to be printed as a real number and inserted into the print stream where the format `%f` occurs. Note the word kilometres in the first argument is not treated as a variable but as an English word since it is inside the double quotes " ".

Now that we have an idea of what the main elements of a C program are it is time to study the language in more detail. C like other languages has an alphabet and rules. These rules are used for making words that can be used as variables and also for putting together variables, keywords and punctuation to make correct, or legal, programs. These rules are the ***syntax*** of the C language. The program that checks whether we have obeyed these rules while writing our program is the compiler. If we have not followed these rules while writing our program the compiler will tell us that there is a **syntax error** in our program and stop. If there are no errors the compiler produces an executable file, which can then be run.

13.2 Alphabet of the C language

A legal C program is constructed using characters. Among the characters that can be used in a program are the following:

Lowercase letters	a b c .... z
Uppercase letters	A B C .... Z
Digits	0 1 2 3 4 5 6 7 8 9
Others	+ - * / = ( ) [ ] < > ' " ! @ # $ % & _ | \ . , ; : ?
White space	blanks, newline, and tab

The character shown above can be combined to form six kinds of words: *keywords*, *variables* (or *identifiers*), *constants*, *string constants*, *operators*, and *punctuators*. These six kinds of words are also called ***tokens***. When the compiler reads a program it first tries to find all the tokens in the program. Let us look at some line of our previous program shown in **Fig 3.1** and try to identify the tokens in it.

Lexical Analysis of the Program

❑

The character “{“ is a punctuator used to separate different tokens. Other punctuators are “,” , “;” and white space (blanks, newline, and tab). Whenever the compiler sees a punctuator it knows that a token has ended.

- `int miles;`

  There are two tokens and two punctuators in this statement. `int` is a *keyword* and `miles` is an *identifier*. The compiler uses the white space between `int` and `miles` to distinguish the two tokens, we cannot write

  ```
  intmiles;
  ```

  the “;” at the end tells the compiler that a statement has ended.

- `kilometres = 1.609 * miles;`

  The characters = and * are operators. White spaces are not need by the compiler in this statement since it can distinguish the tokens without them. White spaces have been put in only to make it more readable. We could have written the statement as

  ```
  kilometres=1.609*miles; or kilometres  =  1.609  *  miles ;
  ```

- `printf("\n300 miles equals %f kilometres.\n\n",kilometres);`

  The name `printf` is an identifier and the parentheses “`( )`” following it tell the compiler that it is a function.

Let us now study all the six forms of tokens in detail.

13.2.1 Keywords

Keywords are words that have been predefined by the inventors of the C language. They have a certain meaning as individual tokens in C. They cannot be redefined or used in other ways. Thus we must never try to use these Keywords to identify our own variables or function names.

C Keywords

```
auto       do       goto       signed    unsigned
break      double   if         sizeof    void
case       else     int        static    volatile
char       enum     long       struct    while
const      extern   register   switch
continue   float    return     typedef
default    for      short      union
```

13.2.2 Variables / Identifiers

Variables are words that are made up of a sequence of letters, digits, and the character _, which is called the underscore. The first character of an identifier must be a letter or an underscore. Some example of legal identifiers are

```
a
_sno
total
```

The following are not legal identifiers

`ille#gel`	the charater # is not allowed in a variable name
`420var`	variables can not start with a digit
`plus-`	_ is allowed but not –

The C language almost always distinguishes between the lower- and uppercase letters. Thus the variable `ABC` is different from the variable `abc`. Some identifiers have already been defined in the C language system. For example `printf` is defined in the `stdio.h`. The identifier `main` is also special, C programs always begin execution at `main`.

On most standard C language systems the length of an identifier can be between 1 to 31 characters. It is considered a good programming practice to give meaningful names to identifiers, according to their use in the program. For example if you write a program that automatically determines the total price of items bought, you should have identifiers like `total`, `no_of_items` and `price`, so that a statement like

```
total = no_of_items * price;
```

would immediately indicate what you are trying to do.

13.2.3 Constants

We used the number 1.609 in the program at the beginning of this chapter. This is an example of a floating constant. Numbers such as 0 and 87 are examples of integer constants. C also has character constants e.g. ‘x’, ‘y’, ‘+’. Character constants must be written between single quotes.

13.2.4 String Constants

Example of string constants are “`My first C program`” and “`123`”. A sequence of characters surrounded by double quote marks is a string constant or a string literal. The compiler considers a string constant as a single token. If you want to include the character “ itself in the string constant you must precede it by a backslash \ character. For example the string “**This is a string with a double quote \” inside it**” will be interpreted by the compiler as **This is a string with a double quote ” inside it**

13.2.5 Operators and Punctuators

Example of operators include the following

```
+ - * / ^
```

Which represent the arithmetic operations of addition, subtraction, multiplication, division and exponentiation respectively. In the C language the parentheses that follow a functions name are also treated as an operator. They operate on the identifier preceding them, to tell the compiler that it is a function name.

Examples of punctuators include parentheses, braces, commas, and semicolons. In the following statements parentheses are treated differently according to the context.

```
printf("My first C program");
a = 17 * (b + c);
```

In the first statement the parentheses are used as an operator on the identifier `printf` to tell the compiler that it is a function. In the second statement they are used as punctuators. You should remember that both operators and punctuators are considered as tokens by the compiler and along with white space, they are used to separate language elements.

13.3 Comments

C is a portable language. This portability means that the program written by one person are used and read by hundred of other programmers. It is very difficult to find out how a program, written by someone else, works by reading its source code. In fact programmers find that they even have difficulty reading and understanding programs they wrote themselves sometime earlier. Comments are inserted in the program to help explain, to anyone reading the code, how the program works. To put a comment in the program place it between the delimiters /* and */. /* Marks the beginning of the comment and */ marks the end. Comments are not tokens. The compiler changes each comment into a single blank space before converting the program to machine language. Therefore comments do not increase the size of the executable program. Some example of comments are given below

```
/* example of a simple comment */

/* example of a comment that
   spans two lines                        */
```

You should write comments whenever you think that it will help in reading and understanding the program and remember there is no need to make your comments short since comments are not included in the program code that it compiles.

Exercises

1. What are tokens and keywords? Is main a keyword?

2. Which of the following are invalid identifiers and why?

   _420a 4_20a reciept# am_I_valid float

3. Find out if nested comments (comments inside comments) are allowed in C. Try to put the following comment in your program.

   /* Let us C /* if this */ works */

4. Why does the comment size have no effect on the executable program size?

5. Can you see what the following code is trying to do? Try to compile and run this code. What does the output tell you about the compiler? What is the difference between the way you see a program and the way compiler sees it?

```
#include<stdio.h> int main(void){printf(“My first\
C program\n”);return 0;}
```

Chapter 14

Data Types

14.1 Declarations

In C a variable must always be declared before it can be used in a program. The compiler will give an error if an undeclared variable is used in a program. The following piece of code shows a couple of ways to declare variables:

```
#include <stdio.h>

int main(void)
{
int a, b, c;                    /* a, b, and c declared of data type
                                      integers                   */
float  total, price;            /* total and price declared of data
                                   type real                     */
.
.
.
```

Declaring variables is simple: type the keyword for the data type of the variables and then a comma separated list of the variable names followed by a semicolon. The compiler uses these declarations in two ways

- First, it uses the declaration to determine how much memory to reserve for each variable. Different types of data require different amount of memory. Recall that the most commonly used unit for memory is called a 'byte'. One byte of memory can use any number between minus128 to plus 127. In most personal computers available today 2 bytes are used to store an integer number, whereas 4 bytes are used to store a decimal number.
- Secondly, the compiler needs to know the data type of a number before it can determine how to carry out mathematical operations such as '+', '-', multiply or divide etc. on that number.

The C language provides programmers with 12 fundamental data types. We will however only discuss 5 most commonly used i.e. ***char***, ***int***, ***long***, ***float*** and ***double***.

14.2 The Data Type `char`

We know that computers store binary numbers in the main memory. So how do we store a character like 'a' in memory? To store characters in memory we associate each character of the alphabet with an integer. For example most machines use an association scheme called ASCII (American Standard Code for Interchange of Information). ASCII

Code associates the number 65 with the character ‘A’, the number 97 with the character ‘a’, the number 66 with character ‘B’, number 98 with character ‘b’ and so on; until we get to the characters ‘Z’ or ‘z’ which are associated with the numbers 90 & 122 respectively (see exercise 4 of chapter 8). If we want to store the character ‘a’ in memory we tell the compiler to reserve a memory space and mark it as of the type `char`. This is done by declaring a variable of type `char`. When we assign that variable a value of ‘a’ the compiler knows that the integer value associated with character ‘a’ is to be put into the memory location associated with the variable. Therefore the value 97 gets stored in the memory location. Similarly if we program the computer to display the value of the `char` variable it knows that the number 97 stored in the memory location is to be displayed as ‘a’. The following program explains this concept practically

```
/* Storing and displaying characters */

#include <stdio.h>

int main(void)
{
    char c;

    c = ‘a’;

    printf(“%c \n”, c);
    printf(“%d \n”, c);

    return 0;
}
```

Analysis Of the Program

- `char c;`

  This line causes the compiler to reserve memory space and associate the variable name c with it. The space reserved will be 1 byte long. Each variable of type `char` is assigned one byte of memory space.

- `c = ‘a’;`

  This is an assignment statement that stores the character ‘a’ in the variable c. However the compiler knows that the character ‘a’ is associated with the number 97 (it uses the ASCII code to look up the number associated with the character ‘a’) therefore it makes sure that when the program executes the value 97 (in its binary form) gets stored in the memory location associated with the variable c.

- `printf(“%c \n”, c);`

  This print statement prints the value stored in the variable c. We pass the format in the first argument to tell it to print the value of c as a character. When this statement executes the computer sees that the value stored in the memory location is 97 and that its character form is 'a', it therefore displays the character 'a' on the screen.

☐ `printf("%d \n", c);`

This print statement prints the value stored in the variable c also, but this time we pass the format `%d` in the first argument to tell it to print the value of c as an integer. When this statement executes the computer sees that the value stored in the memory location is 97 it therefore displays it as an integer in decimal format 97 on the screen.

It is important to realize that a character representing a number e.g. '2' is not equal to number 2 itself. For example, when you store '2' in a character variable the computer looks up the number associated with the character '2' in the ASCII table and finds 50, it then stores the number 50 in the memory location.

14.3 The Data type `int`

The set of numbers 0, 1, 2, 3, ….., and their negatives -1, -2, -3, …. is called the integers. When we want to store integer values in our variables we declare the variable to be of type `int`. On most personal computers the compiler reserves two memory locations (bytes) to store the value of the variable(see exercise 1 at the end of this chapter). Recall that the compiler reserves on one memory location for a variable of the type `char`. Using two locations of memory the computer can store values between -32768 and 32767 in a variable of type `int`. Thus there is limit on the value you can store in an `int` variable. Now suppose that you write down the following program on a machine that assigns two memory locations for an `int` variable.

```
#include <stdio.h>

int main(void)
{
    int a, b, c;

    a = 30000;
    b = 30000;
    c = a + b;
...
```

The compiler will compile this program into an executable file. But when you run this program the value of the variable 'c' will not be set equal to 60000 because a variable of type `int` can only store values up to 32767. This condition is called an ***integer overflow***. Typically, when an integer overflow occurs, the program continues to run, but it produces results, which are not correct. It is the job of the programmer to make sure that the value stored in an `int` variable does not exceed the limit at any stage of the calculation.

14.4 The Data type `long`

The C language provides the data type `long` for cases in which large integers numbers are to be stored. The compiler reserves four memory cells for a variable of type `long`. Using the type `long` we can store values between -2147483648 and 214783647 in variables. If you want to store integers that are even bigger than this you will have to combine several variables of type `int` or `long` through programming to do the job.

14.5 Data types `float` and `double`

These two data types are used to store real numbers. Real numbers are those numbers that have a decimal point in them e.g. 0.1, 0.001, 3.1423. The number 4 is an integer but 4.0 is a real number. Real numbers can also be written in an alternative way called exponential notation. For example

$$\mathbf{123.4567} = 0.1234567 \times 1000 = 0.1234567 \times 10 \times 10 \times 10 = \mathbf{0.1234567 \times 10^3}$$

Similarly

$$\mathbf{0.000123456} = 0.123456 \times 1/1000 = 0.123456 \times 10^{-1} \times 10^{-1} \times 10^{-1} = \mathbf{0.123456 \times 10^{-3}}$$

As you can see from the preceding two examples we can write all real number in the following standard form

$$\underbrace{0.d_1d_2d_3\ldots d_m}_{\text{precision}} \times \underbrace{10^n}_{\text{range}}$$

Figure 14.1: Structure of a real number

We define ***precision*** as the number of digits after the decimal point in the above standard form and ***range*** as the exponential power of 10.
For the number $\mathbf{0.1234567 \times 10^3}$ we have

$d_1 = 1$, $d_2 = 2$, $d_3 = 3$, ... $d_7 = 7$

$m = 7$ and $n = 3$

precision = 7 range = 10^3

Similarly for $\mathbf{0.123456 \times 10^{-3}}$ we have

$d_1 = 1$, $d_2 = 2$, $d_3 = 3$, ... $d_7 = 7$

$m = 6$ and $n = -3$

precision = 6 range = 10^{-3}

The possible values that a floating type variable can be assigned are described in terms of precision and range. A number declared of type `float` on most machines has a precision of 6 and a range of 10^{-38} to 10^{+38}. Suppose we declare a variable to be of type `float` and then write the following statement in the program

```
f = 123.456123456;
```

if we write down the number in our standard form we get

123.456123456 = 0.123456123456 x 103

the precision for this number is 12 and the range is 10^3. A variable of type `float` can store the range 10^3 but it cannot store a precision of 12. Therefore only the first 6 digits of the precision will be stored and the computer will store the following number in memory

0.123456×10^3

The type `double` is also used to store real number just like `float`, but it has a precision of 15 digits and a range of 10^{-308} to 10^{+308}.

We will not go into the details but there are two more points that you should be aware of

- Not all real numbers are presentable using the type `float` on a computer. The main reason for this is that computers store numbers in binary form and some real numbers don't have an exact binary representation.
- When you perform operations such as multiplication using `float` variables the result might not be exact. The reason is again that the computer uses binary numbers when it actually carries out the operations and due to the limited representation and space available the results may not come out to be exact.

14.6 Type Qualifier `const`

The value of a variable can be changed in a program. However there are situations in which you do not want the value of a variable to be changed once it has been initialized to some value. The C language provides the type qualifier `const` for this purpose. Consider the following line of code:

```
const int AREA = 2048;
```

When the compiler reads the above line it stores the value 2048 in the variable `AREA` and then marks it as a constant. If you try to store a new value in this variable the compiler will mark it as an error.

Exercises

1. To find out the number of memory spaces that a variable of a certain type is assigned by the compiler we can use the ***sizeof*** operator. The following program shows you how to use the sizeof operator to find out precise

information about the storage requirements for the fundamental data types on your computer. Run the program on your computer and study the results.

```
/*Print the size of the important data types in C*/

#include <stdio.h>

int main(void)
{
    printf(“The size of  some data type in bytes is as
follows:\n\n”);
    printf(“     char:%3d byte  \n”, sizeof(char));
    printf(“      int:%3d byte  \n”, sizeof(int));
    printf(“     long:%3d byte  \n”, sizeof(long));
    printf(“    float:%3d byte  \n”, sizeof(float));
    printf(“   double:%3d byte  \n”, sizeof(double));
    return 0;
}
```

2. What is *integer overflow*? The following program declares a variable test_var of type int, initializes it to 1 and then keeps on incrementing it. Each time the variable is incremented, the program prints out its value. Compile and Run the program on your computer. Is there a maximum value for the variable test_var that is printed? What happens after that?

```
#include <stdio.h>

int main(void)
{
   int i=1;

   while (1)
   {
      printf(“The value in i is : %d\n”, I);
      i++;
   }
return 0;
}
```

Note that the program has an infinite loop in it. You will have to interrupt the program execution by pressing control-c to stop it.

3. When a variable is declared the complier reserves space for it in the memory. Most compilers however do not initialize this memory space to some value. Write a program to print the value of different variables that have been declared but have not yet been assigned any value in the program. Run the program several times. Do you get the same values each time?

4. Variables with data types studied in this chapter store a single value. If you want to store a list of values e.g. the exam scores of a class these data types

cannot be used. The C language provides the ***array*** data type for this purpose. The following program declares an array of type int (only integer values will be allowed in the array) and then stores the exam marks of five students in the array before printing them out.

```
#include <stdio.h>

int main(void)
{
int marks[4];         /* This declaration makes the compiler reserve five
                         spaces in memory which have identifiers a[0], a[1],
                         a[2], a[3] and a[4]                              */

a[0] = 94;            /* Storing value in the first space. Note that counting
                                                              starts at 0 */
a[1] = 57;            /* Storing value in the second space.*/
a[2] = 83;            /* Storing value in the third space.*/
a[3] = 77;            /* Storing value in the fourth space.*/
a[4] = 91;            /* Storing value in the fifth space.*/

           /* Printing the values in the array */
printf("Marks for student no 1: %d\n", a[0]);
printf("Marks for student no 2: %d\n", a[1]);
printf("Marks for student no 3: %d\n", a[2]);
printf("Marks for student no 4: %d\n", a[3]);
printf("Marks for student no 5: %d\n", a[4]);

return 0;
}
```

Change this program so that it stores the marks as real number e.g. 88.8. Initialize the array elements to some floating point values and then print these values.

Chapter 15

Operators

Operators are used with variables and constants to form expressions. Some examples of expressions are

```
a + b
a * 5 + b
```

In the expression a + b, + is the operator while a and b are operands. Operators like / (the division operator) are called binary operators because they always take two operands. The operators ++ and -- take on only one operand and are therefore called unary operators. The + and - operators work with either one or two operands. We will study the following kind of operators arithmetic, assignment, relational and logical.

15.1 Arithmetic Operators

Arithmetic Operators			
Operator	**Name**	**Description**	**Example**
++	Increment	Increases the value of the operand by 1	++a, a++
--	Decrement	Decrease the value of the operand by 1	--a, a--
*	Multiplication	Multiply two numbers	a * b
/	Division	Divide two numbers	a / 2
%	Modulus	Find remainder of a division	10 % 3
+ (binary)	Addition	Add two numbers	a + b
- (binary)	Subtraction	Subtract two numbers	c – d
+ (unary)	Positive	Marks a number as positive	+b
- (unary)	Negative	The negative of the operand	-b

Figure 15.1: Arithmetic Operators

The * , +, / and - are the usual arithmetic operators you have studied in mathematics. The following example should make their usage clear.

Declarations and initializations	
i = 1, j = 2, k = 3	
Expression	**Value**
i + j	4
k – l	2
j * k	6
j + k – i	4

Figure 15.2: Expressions with arithmetic operators

Note that the an expression like `j + k` has a value which can be assigned to other variables e.g. `i = j + k;`. To Calculate the expression `j + k - i`, first the expression `j + k` is evaluated and a value of 5 is assigned to it, then the expression `5 - i` is evaluated and the value of 4 is assigned to the whole expression `j + k + i`. The modulus operator is used to calculate the remainder of a division between two integers. The expression `10 % 3` has a value equal to 1. The expression `3 % 5` has a value of 3 since 3 is not at all divisible by 5. Similarly `0 % 4` has a value 0.

15.2 Increment and Decrement Operators

The operators we have studied till now have been binary operators; they operate on two values e.g. in `j + k` the `+` operator operates on j and k. The increment operator `++` and decrement operator -- are unary operators they are applied to single variables. Note that other operators do not change the value of the variables they are applied to, but the increment and decrement operators increase or decrease the value of the variable they are applied to by 1. There are two ways to apply these operators, prefix `++i` or postfix `i++`. The difference is explained below

```
int a, c;
a = 0;
c = 0;

c = a++;
printf(“%d %d\n”,a, c);          /*  1 and 0 is printed  */
```

When this program is executed and we reach the statement `c = ++a;` the current values of a, b and c are all zero. When we evaluate the expression `c = ++a`, the following happens

- The expression `a++` is evaluated. The expression `a++` has as its value the current value of a, therefore the expression `a++` has the value zero.
- The value of the variable `a` is incremented by 1, therefore a now has the value 1.
- The value of `a++` is assigned to the variable c, therefore c now has the value 0.

Now suppose we have the following program

```
int a, c;
a = 0;
c = 0;

c = ++a;
printf(“%d %d\n”,a, c);
                                /*  1 1 is printed  */
```

In this case the following steps take place

- The value of the variable `a` is incremented by 1, therefore a now has the value 1.

- The expression `++a` is evaluated. The expression `++a` has as its value the current value of a, therefore the expression `++a` has the value 1.
- The value of `++a` is assigned to the variable `c`, therefore c now has the value 1.

The decrement operator `--` is used in a similar way. If we just have the statements `++a;` or `a++;` in our program then their use is equivalent. For example consider the following program

```
int a, c;
a = 0;
c = 0;

++a;
printf("%d %d\n",a, c);          /*  1 and 0 is printed  */
```

If we had used the statement `a++;` instead of `++a;` the function `printf` would still output the same values.

15.3 The Assignment operator

The equal sign = is treated as an operator in C. We have already seen = being used in expressions like

```
a = 0
c = a + b
```

These assignment expressions have the following form.

variable = *right_side*

When an expression like `a + b` with the operator `+` is evaluated, the values of `a` and `b` are added together and the whole expression '`a + b`' has this sum as its value. Similarly when an assignment expression is evaluated, the value of the *right_side* is assigned to *variable*, and that value becomes the value of the assignment expression '*variable* = *right_side*' as a whole. For example when the expression a = 2 is evaluated, the variable a is assigned the value 2 and then the whole expression 'a = 2' is assigned the value 2. Treating the equal sign = as an operator makes multiple assignments possible. Consider the following statement

```
a = b = c = 0;
```

All the variables in this statement get 0 as their value. This happens because first the expression `c = 0` is evaluated, which makes the value of c equal to 0 and the whole expression '`c = 0`' is then assigned the value 0. Next `b = c = 0` is evaluated, but the value of '`c = 0`' is 0 therefore b gets the value 0 and the whole expression '`b = c =0`' gets the value 0. Finally the same thing happens with a, which is assigned the value 0, and the expression '`a = b = c = 0`' also gets the value 0 as a whole.

Another assignment statement that sometimes confuses beginners has the form

```
a = a + 1;
```

Students sometimes confuse this with an expression from algebra. In algebra '`a = a + 1`' is not possible. The C language expression '`a = a + 1`' is however different and is evaluated as follows

- the value of the expression `a + 1` is evaluated by adding 1 to the current value of `a`.
- The value calculated in step 1 is assigned to the variable a
- The whole expression '`a = a +1`' is then assigned this value from step 1

Thus if the current value of the variable a is 0, after the execution of the statement `a = a +1;` the value of `a` becomes 1.

15.4 Relational Operators

The relational and equality operators

```
<    >    <=    >=    ==    !=
```

are all binary, and are used to form expressions like `a < b`, `c == a` etc. The values of these expressions are either the `int` value 0 or the `int` value 1. The value int 1 assigned to these expressions represents ***true*** while the value int 0 represents ***false***. Therefore a relational expression is assigned a value of 1 if it is true otherwise it get a value of 0. The reason for this representation is that in the C language *true* can be represented by any nonzero value, while *false* can be represented by any zero value. If there are two variables a and b three possibilities exist

- Value of a is less than the value of b
- Value of a is equal to the value of b
- Value of a is greater than the value of b

The following table shows what values the relational expressions take for the above three cases. For example, if a=1 and b=2 then a is less than b and the value of a > b is 0.

Relation	**Example**	**a < b**	**a > b**	**a <= b**	**a>=b**	**a==b**	**a!=b**
a is less than b	`a=1, b=2`	1	0	1	0	0	1
a is equal to b	`a=5, b=5`	0	0	1	1	1	0
a is greater than b	`a=10, b=2`	0	1	0	1	0	1

Figure 15.3: Relational expressions and their values

The following program prints the value of some of these expressions.

```
#include <stdio.h>

int main(void)
{
   int a=1, b=2;

   printf(“if a=1 and b=2 the value of a<b is %d\n”, (a<b));
   printf(“if a=1 and b=2 the value of a>b is %d\n”, (a>b));
   printf(“if a=1 and b=2 the value of a<=b is %d\n”, (a<=b));
   printf(“if a=1 and b=2 the value of a>=b is %d\n”, (a>=b));
   printf(“if a=1 and b=2 the value of a==b is %d\n”, (a==b));
   printf(“if a=1 and b=2 the value of a!=b is %d\n”, (a!=b));

   }
return 0;
}
```

We will make use of relational expression when we study the conditional and loop statements.

15.5 Logical Operators

C has three logical operators `!`, `&&` and `||`. When these operators are applied to expressions either the int value 1 or the int int value 0 is produced as a result.
The logical negation operator ! negates the truth value of an expression. Recall that all expressions that evaluate to a nonzero value have a truth value *true* and expressions that evaluate to zero have a truth value *false*. If an expression has a nonzero value (true), then its negation will yield the int value 0 (false). If the expression has value zero (false), then its negation will yield the int value 1 (true). This show in the table below

Value of Expression expr	Value of Expression !expr
Zero	1
Nonzero	0

Figure 15.4: Values of negated expressions

The binary logical operators && and || also act on expressions and yield either the int value 0 or the int value 1. Examples of usage are

```
a && b
c || d
!(a < b) && c
(a > b) && (2 * a + 4)
```

The logical expressions involving these are evaluated according to the following rules

Value of expr1	**Value of expr2**	**Value of expr1 && expr2**	**Value of expr1 || expr2**
zero	Zero	0	0
zero	Nonzero	0	1
nonzero	Zero	0	1
nonzero	Nonzero	1	1

Figure 15.5: Logical expressions and their values

15.6 Operator precedence and associativity

Let us suppose that we are asked to find the value of the expression `1 + 2 * 3`. There are two ways to evaluate this expression

- Multiplying 2 and 3, and then adding 1 to the result. This gives us 7 as the value of the expression.
- Adding 1 and 2, and then multiplying the result with 3. This gives us 9 as the value of the expression.

The meaning of the expression `1 + 2 * 3` is therefore not clear. To get rid of this ambiguity the C language defines rules of precedence and associativity. These rules tell the compiler exactly how expressions are to be evaluated. For example in C the multiplication operator * has higher precedence than the addition operator + therefore when the expression `1 + 2 * 3` is evaluated the operation of multiplication between 2 and 3 will be performed first and the result will then be used in the addition operation with 1. If we want the addition to be performed first, we can use the following expression

```
(1 + 2) * 3
```

The parentheses `( )` have higher precedence than multiplication therefore the expression inside the parentheses is performed first. Programmers also use parentheses to make mathematical expressions more readable. For example the `1 + 2 * 3` is equivalent to the expression

```
1 + (2 * 3)
```

The parentheses help us visually identify the expression `(2 * 3)` as the one whose value is to be added to 1. Now consider the expression

```
4 * 5 / 10
```

The binary operators * and / have the same precedence, therefore using only the precedence rules the compiler cannot decide which of the two operations we want performed first. This is where the associativity rules come in. The associativity rules tell the compiler how to perform the evaluation if the expression contains operators that have equal precedence. The associativity rule for * and / is "left to right", the compiler will therefore start performing operation from the left side and move towards the right side. In the expression `4 * 5 / 10`, `4 * 5` will be evaluated first and the result will then be divided by 10. We can write the equivalent expression as

```
(4 * 5) / 10
```

The following table gives the precedence and associativity of the operators in C.

Operator Precedence and Associativity	
Operators	**Associativity**
() ++ (postfix) -- (postfix)	left to right
++ (prefix) -- (prefix)	right to left
/ %	left to right
+ -	left to right
=	right to left

Figure 15.6: Operators and their precedence and associativity

The operators in each row have higher precedence than the operators in the row below. The assignment operator has the least precedence.

Exercises

1. Give equivalent logical expressions of the following without negation:

```
!(a > b)                     !(a <= b && c <= d)
!(a+ 1 == b+1)               !(a < 1 || b < 2 && c < 3)
```

2. Complete the following table. Use parentheses to give the equivalent expressions

Declarations and initializations

int a = 1, b = 2, c = 3
double x = 1.0;

Expression	Equivalent expression	Value
a > b && c < d		
a < ! b || !!a		
a + b < ! c + c		
a – x || b * c && b /a		

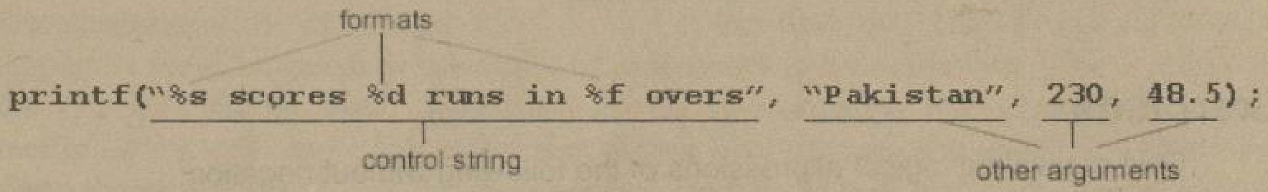
Chapter 16

Input / Output

We have seen how to output values from the program using the `printf()` function. Another function the is provided by the C language is `scanf()`. This function is used to input a value into the program. In this Chapter we discuss these two functions in detail.

16.1 `printf()` the output function

The `printf()` function not only outputs values given to it, it can also change the format and appearance of the values according to the programmers specifications. The first argument to the `printf()` function is special, it is called the ***control string***.



```
printf("%s scores %d runs in %f overs", "Pakistan", 230, 48.5);
```

Figure 16.1: Parts of the printf() function.

The control string may contain strings starting with the `%` symbol, these are called ***conversion specifications***, or ***formats***. Characters in the control string that are not part of a format are placed directly in the output. In the example above the empty spaces and the characters in the words 'scores', 'runs', 'in', and 'overs' are printed as they are without any change. The expressions in the other arguments are evaluated and converted according to the formats in the control string and then printed. In our example of Figure 16.1 the argument "Pakistan" is converted according to the corresponding format `%s`, `230` is converted by `%d` and `48.5` is converted by `%f`. The following table shows different formats and how their corresponding arguments get printed.

formats	Corresponding argument printed as
c	a charater
d, I	a decimal integer
f	a real number
s	a string

Figure 16.2: printf() formats.

If we run a program containing the statement in Figure 16.1 we will get the following output

```
Pakistan scores 230 runs in 48.500000
```

Note that the format `%f` converted `48.5` into `48.500000` before printing. This happened because the default action of `%f` prints a real number with six digits to the right of the decimal point. To change the default behaviour of a format we can place various characters between the `%` sign and format character. This is shown in Figure 16.3 below

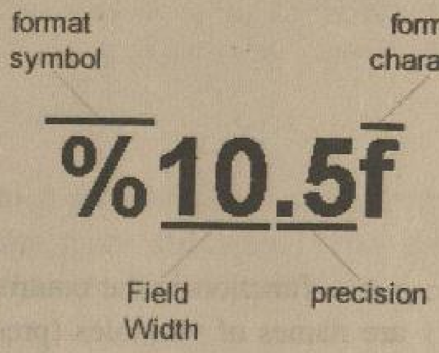


Figure 16.3: Various fields of the printf() format.

Between the `%` that starts a format and the format character that ends it, there may appear in the following order

- An optional positive integer that specifies the minimum field width of the converted argument. The place where an argument is printed is called its ***field***, and the number of spaces used to print an argument is called its ***field width***. If the converted argument has fewer characters than the specified field width, then it will be padded with spaces on the left or right, depending on whether the converted argument is right- or left-adjusted. If the converted argument has more characters than the specified field width, then the field width will be extended to whatever is required. If the integer defining the field width begins with a zero and the argument being printed is right-adjusted in its field, then zeros rather than spaces will be used for padding.
- An optional precision, which is specified by a period followed by a nonnegative integer. For the `d` and `i` conversions, it specifies the minimum number of digits to be printed. For the `f` conversion, it specifies the number of digits to the right of the decimal point. For the `s` conversion it specifies the maximum number of characters to be printed from a string.

16.2 `scanf()` the input function

`scanf()` works in a way similar to the `printf()` function except that it places values read from the input, in this case the keyboard, into the variables in the program. The following code shows an example of the use of `scanf()`

```
#include <stdio.h>

int main(void)
```

```
{
    int miles;
    char c;
    float kilometers;

    scanf(“%d %c”, &miles, &c);

    kilometers = 1.609 * miles;
     printf(“%d %c %f kilometers.\n”, miles, c, kilometers);
```

Analysis of the Program

- `scanf(“%d %c”, &miles, &c);`

  The first argument to the `scanf()` function is the control string `"%d %c"`. The other arguments to `scanf()` are names of variables (preceded by a `&` sign) and they match the formats in the control string. In our example `%d` matches `&miles` and `%c` matches `&c`. The control string has two formats `%d` and `%s`. Therefore `scanf()` will try to input two values from the input device. `%d` is the format for a decimal integer value, just like in `printf()`. Since it corresponds to the first value read, `scanf()` will read the first value entered as a decimal integer and store it in the variable miles. `%c` is the format for a character value. The second value will be read as a character and stored in the variable `c`. Note that there is a space between `%d` and `%c` in the control string. When you input the values you will also have to put in a space between the two values entered. When you run the program, the computer will wait for you to enter the input from the keyboard. If you enter the following values

  ```
  300 =
  ```

  that is `300`, a space and the `=` sign (press enter after you have entered these values), `scanf` will place a value of `300` in the variable `miles` and the character = in the variable `c`.

- `printf(“%d miles %c %f kilometers.\n”, miles, c, kilometers);`

  This statement will output the following line

  ```
  300 miles = 482.7 kilometers
  ```

The formats for `scanf()` are described in the Figure 16.4 below

format	matched character in the input	Corresponding argument
c	any character, including white space	&c, where c is any variable of type char
d, i	a decimal integer	&d, where d is any variable of type integer e.g. &miles
F	a real number	&f, where f is any variable of type float e.g. &price

Figure 16.4: scanf() formats

The control string may contain

- White space, which matches optional white space in the input stream.
- ordinary nonwhite space characters, other than `%`. Each ordinary character must match the next character in the input stream.

Exercises

1. Certain characters can be represented in character and string constants by ***escape sequences*** like \n; these sequences look like two characters, but represent only one. \n represents the character *newline*. As we have seen, when we print the character \n it causes the output to advance to the beginning of the next line. Other escape sequences are given below.

\a	alert (bell) character	\\	backslash
\b	backspace	\?	question mark
\f	formfeed	\’	single quote
\n	newline	\”	double quote
\t	horizontal tab	\v	vertical tab

   Write programs and experiment with these escape sequences to find out there effect on the output

2. Experiment to find out what happens when printf() contains \c, where c is some character not listed in the escape sequences in question 1 above.

3. Find out whether %f format truncates or rounds a real number when you try to print fewer decimal characters then represented by the variable.

Chapter 17

Decision Statements

Decision-making is an important part of life. Some of the decisions we make depend on whether certain conditions are true or false. For example at a traffic signal if the signal is red you stop otherwise you keep going. We can also program computers to perform certain actions only when certain conditions are true. This is done through the `if` and `if-else` statements.

17.1 The `if` Statement

The general form of an if statement is

if (*expression*)
    statement

If the *expression* is nonzero (true), then *statement* is executed otherwise the *statement* is ignored and the computer tries to execute the next statement in the program. Suppose you want to print the message “Ah! 7 is considered a lucky number” only when a certain variable has the value 7. You can do this using the `if` statement as shown below

```
#include<stdio.h>

int main(void)
{
int num;

scanf(“%d”, &num);

if (num == 7)
    printf(“Ah! 7 is considered a lucky number\n”);

return 0;
}
```

The *expression* is usually a relational, or logical expression, but it can be an expression of any type that evaluates to a zero or a nonzero value. A common programming error is to place the expression `a = b` instead of `a==b` in the `if` statement. For example if you write the program in the last example with the expression `(num = 7)` instead of `(num == 7)`

```
int num;

scanf(“%d”, &num);

if (num = 7)
```

```
    printf("Ah! 7 is considered a lucky number\n");

return 0;
```

The program will print "`Ah! 7 is your lucky number`" no matter what number you enter. The reason for this is that the expression `num = 7` is an assignment expression with a value of 7 (see section 5.3 on the assignment operator) and this is a non zero (true) value which always causes the `printf()` statement to execute.
A ***compound statement*** is a series of declarations and statements surrounded by braces. For example the following compound statement contains two statements in it

```
{
var1 = 3;
printf("value of variable 1 is %d", var1);
}
```

When a group of statements are to be executed when a condition is true we use the compound statement in the `if` statement. The following code gives an example

```
if (j < k) {
    min = j;
    printf("j is smaller than k\n");
}
```

Because and an `if` statement is itself a statement, it can be used as the statement part of another `if` statement, as the following code shows

```
If ( a == 1)
    If (b==2)
         printf("a < b");
```

The above statement first checks whether the variable `a` has value 1, if it has then it checks if the variable `b` has value 2.

17.2 The `if else` Statement

The if-else statement is closely related to the `if` statement. it has the general form

if (*expression*)
    statement1
else
    statement2

If expression is nonzero, then *statement1* is executed and *statement2* is ignored; if expression is zero, then *statement1* is ignored and *statement2* is executed. Consider the following example

```
if (x < y)
```

```
    min = x;
else
    min = y;
```

If `(x  < y)` is true, then `min` will be assigned the value of `x`, and if it is false, then `min` will be assigned the value of `y`.

17.3 Dangling else

When an if-else statement is used as the statement part of another if statement ambiguity arises. Consider the following code

```
If (a == 1)
If (b ==2 )
      printf("Statement1");
else
      printf("Statement2");
```

It is not clear from the syntax which `if` the `else` part is associated with. This is known as the ***dangling else problem***. The rule to resolve this is that an `else` attaches to the nearest `if`. Therefore the else belongs to the second if statement. We can format the code to emphasize this point.

```
If (a == 1)
   If (b == 2 )
          printf("Statement1");
   else
          printf("Statement2");
```

17.4 The Conditional Operator

The conditional operator `?:` takes three operands in its expressions, it is therefore a ternary operator. The general form is

expression1 ? *expression2* : *expression3*

expression1 is evaluated first. If it is nonzero, then *expression2* is evaluated and that is the value of the conditional expression as a whole. If *expression1* is zero, then *expression3* is evaluated and that is the value of the conditional expression as a whole. Thus a conditional expression can be used to do the work of an `if-else` statement. Consider the code

```
if (x < y)
    x = y;
else
    x = z;
```

The same can be accomplished by using the conditional statement

```
x = (y < z) ? y : z;
```

17.5 The switch Statement

The switch statement is a way of generalizing the if-else statement. It has the general structure.

```
switch(expression)
{
case label1:
     statement1;
     break;
case label2:
     statement2;
     break;
     .
     .
     .
     .
default:
     default statement;
     break;
}
```

The switch statement is evaluated using the following steps

- Evaluate the switch *expression*.
- Check the case labels for a value that matches the value of the expression calculated in step 1. If a matching case if found execute the associated statement. If a match is not found, go to the default label, or, if there is no default label, terminate the switch.
- Leave the switch statement when a `break` statement is encountered, or terminate the switch by "falling off the end."

The *expression* in the parentheses following the keyword `switch` must be of integral type. Each case label is an integer-valued constant. All case labels must be different. Typically, the last statement before the next `case` or `default` label is a `break` statement. Whenever a `break` statement is encountered, execution of the switch statement is terminated and the next statement in the program is executed. If a case matches the expression value and it does not contain a `break` statement then the statements of all the remaining cases are executed without matching the case labels with the switch expression. Missing `break` statements are a frequent cause of error in switch statements. There may be at most one `default` label in a switch.

Exercises

1. Find the errors in the following statements

a)
```
if ( c < 7 );
     printf("c is less than 7\n");
```

b)
```
if (c=>7)
    printf("c is equal to or greater than 7\n");
```

2. Write a program that asks the user for two numbers, obtains the numbers and stores them in two variables. The program then prints the greater of the two numbers. If the numbers are equal than it prints the value of any of the numbers.

3. In the following program we are trying to print the value of the variable **label**. What do you think the output of the program is?

```
#include <stdio.h>

main()
{
int label;

label = 1;

 switch (label){
 case 0: printf("label has value 0\n");
 case 1: printf("label has value 1\n");
 case 2: printf("label has value 2\n");
 case 3: printf("label has value 3\n");
 default : printf("label has no matching value\n");
     }
return 0;

}
```

If the program output is not correct, make changes in the code so that the program output is what we want it to be.

Chapter 18
Loops

One of the main reasons we use computers is that they can execute a set of operations repeatedly. Using mechanisms called loops we can run a group of instructions on large amounts of data or even produce data using these instruction. We will study the *for*, *while* and *do* loops in this chapter.

18.1 The while statement

Let us start with an example. This program takes an integer n as input from the user and prints out all the integers from 0 to n.

```
/* program for producing integers in the ranage 0 to n where n
is a given number using the while loop
*/

#include <stdio.h>

int main(void)
{
      int n, i=0;

      printf(“Enter a nonnegative integer n : \n”);
      scanf(“%d”, &n);

      while ( i <= n )
{
      printf(“%d, ”, i);
      i++;
}
      return 0;
}
```

Program Analysis

The while statement has the following form

while (*expression*)
statement

next_statement

When a while statement is encountered in a program the *expression* is evaluated first. If the value of the expression turns out to be nonzero (true), then *statement* is executed. The computer now goes back to the while statement and evaluates the *expression* again. If the expression is still true the *statement* is executed and the control passes to the while statement once more. The computer repeats this process until the expression becomes zero (false). At that point the *next_statement* is executed.

The *statement* associated with the while statement can be a *compound statement*. Therefore to associate a group of statements with the while statement, enclose them between a pair of brackets `{ }`. We have done this in our example above where the while statement has the following form

```
    while ( i <= n )
{
    printf("%d, ", i);
    i++;
}
    return 0;
```

The variable `i` of type `int` whose value has been initialized to zero and the variable `n` holds the value entered by the user. Let us suppose that the user has entered the value `4`, which has now been assigned to the variable `n`. The first time the while statement in executed the expression `(i <= n)` will have the value equal to the expression `(0 <= 4)` because `0` and `4` are the current values of `i` and `n`. This expression has the value *true*, therefore the instructions

```
printf("%d, ", i);
i++;
```

will be executed. The computer will print the number `0` and increment the value of `i`, which will now become `1`. The control will now pass back to the while statement and the expression will now be evaluated as `(1 <= 4)`. This cycle will go on until the value of `i` becomes `5`. The computer will have printed the following values upto this point

```
0 1 2 3 4
```

Now when the control goes back to the while statement the expression `(5 <= 4)` will evaluate to false and the control will pass to the next statement, which is

```
return 0;
```

Notice that if we specify and expression which never becomes false the computer will execute the while statement indefinitely. Therefore care should be taken to check that the condition when the expression becomes false does eventually occur in the program.

18.2 The for Statement

The `for` statement performs the same functions as the `while` statement, only the format is a little different. The general format of the `for` statement is

for (*expression1*; *expression2*; *expression3*) *statement;*

next_statement

The *statement* might be a single statement or a compound statement. The `for` statement is executed as follows. First *expression1* is evaluated. In most programs *expression1* is used to initialize a variable. Then *expression2* is evaluated. If it is nonzero(true), then the statement is executed. *Expression3* is evaluated next and the control passes back to the beginning of the `for` loop. *Expression2* is now evaluated. If *expression2* evaluates to true the statement is executed again and *expression3* is evaluated. This cycle goes on until *expression2* becomes false at which point the control passes to the *next_statement*. Notice that expression1 is only evaluated at the beginning of the first loop, on every subsequent loop it is skipped. We now present the program of previous section using the `for` loop instead of the `while` loop

```
/* program for producing integers in the ranage 0 to n where n
is a given number using the for loop                          */

#include <stdio.h>

int main(void)
{
    int n, i;

      printf("Enter a nonnegative integer n : \n");
      scanf("%d", &n);

      for (i = 0; i <= n; ++i)
{
      printf("%d, ", i);
}
      return 0;
}
```

A major advantage of using a `for` loop instead of a `while` loop is that the statement which control the execution of the loop are all present at the top of the loop and it readily becomes apparent what the program is going to do. This is especially useful when the loops are nested, that is there are loops within loops.

18.3 The do Statement

The `do` Statement makes it easy to implement a loop when the loop statement must be executed at least once. The general form of the do statement is

do
    statement
while (*expression*);

next_statement

When the control passes to the do loop first the *statement* is executed and then the expression is evaluated. If the *expression* value turns out to be true, control passes back to the start of the do loop. This process continues until the *expression* becomes false, at which point the control passes to the *next_statement*. The program for printing integers using a do loop is

```
/* program for producing integers in the ranage 0 to n where n
is a given number using the do loop                          */

#include <stdio.h>

int main(void)
{
    int n, i=0;

    printf("Enter a nonegative integer n : \n");
    scanf("%d", &n);

    do
          {
printf("%d, ", i);
i++;
}
    while ( i <= n );

    return 0;
}
```

Exercises

1. Write a program that prompts the user for to inputs two integers values for the variables *width* and *height*. After obtaining the values for these variables, it prints out a box similar to the one shown in the figure below

```
**********
*        *
*        *
*        *
*        *
*        *
**********
```

Where the number of * in the first and last line is equal to the value of the width variable. The number of lines between the first and last line is equal to the value of the height field.

2. Write a program that prints out the average of all the non-zero integer number input by the user. The user indicates that the input has ended by entering the value 0. A typical input might be
   `10 -4 34 -7 99 0`

   Indicating the all the values preceding the value zero should be averaged.

3. What does the following program print

```
#include <stdio.h>

int main(void)
{
       int i, j;

       for (i = 1;  i <= 10; i++)
              {
              for (j = 1; j <= i;  j++)
                     printf("%d", j);
              printf("\n");
              }
       return 0;
}
```

4. The ASCII character code was introduced in section 4.2. ASCII code associates integer numbers in the range 0-127 with characters. Of these the characters represented by the integers in the rage 32-126 can be printed on the screen. The following program prints the printable ascii characters and the associated integers on the screen. Compile and run the program on your computer.

```
#include <stdio.h>
main()
{
char c;

for (c = 0; c < 127; c++)
printf("%d \t %c\n", c, c);

return 0;
}
```

   Write a program that inputs a character from the user and prints its associated ASCII integer number.

5. Suppose that we have an array with the following eight numbers in it
   `a[0]=7,  a[1]=3,  a[2]=66,  a[3]=3,  a[4]=-5,  a[5]=22,`
   `a[6]=-77,a[7]=2`
   We want to sort the values in this array such that, a[0] contains the smallest values and a[7] contains the highest value. We intend to sort the array in n-1 steps or **passes**, where n is the total number of elements in the array. At the start

of the first pass a[6] is compared with a[7]. If the value of a[7] is less than a[6] we exchange them otherwise we leave them alone as they are. Next we compare a[5] with a[6] and exchange them if a[6] is less than a[5]. Then a[4] is compared with a[5] and so on. The result of the first pass is that the smallest value in the array gets stored in a[0]. In the second pass the same procedure is repeated except that we compare values upto the value in a[1] and the value in a[0] is left alone. After the second pass the next to the smallest value is in a[1]. Since each pass bubbles the next smallest element to its appropriate array position, the algorithm will order the array in n-1 passes. This is show in the following table:

Original array:	7	3	66	3	-5	22	-77	2
First pass:	-77	7	3	66	3	-5	22	2
Second pass:	-77	-5	7	3	66	3	2	22
Third pass:	-77	-5	2	7	3	66	3	22
Fourth pass:	-77	-5	2	3	7	3	66	22
Fifth pass:	-77	-5	2	3	3	7	22	66
Sixth pass:	-77	-5	2	3	3	7	22	66
Seventh pass:	-77	-5	2	3	3	7	22	66

The following code orders the array as described above. Write a program to sort an array and use this code in it.

```
int i, j, tmp;

for (i = 0; i < n - 1; ++i)
      for (j = n-1; j > i; --j)
      if (a[j-1] > a[j])
      {
           tmp = a[j];             /* exchange array values */
           a[j] = a[j-1];
           a[j-1]= tmp;
      }
```

Chapter 19

Functions

19.1 Importance of Functions

Suppose we have been asked to write a program that reads in two integers *x* and *y* from the user, calculates the expression $x^3 + y^3$ and prints it on the screen. For example if we input the number 3, 4 the program should calculate $3^3 + 4^3$ and therefore print 91. A possible program to do this is

```
#include <stdio.h>

int main(void)
{
    int x, y;
long xp, yp, result;

     printf("Enter the values for x and y");
     scanf("%d %d", &x, &y);

     xp = x * x * x;
     yp = y * y * y;

     result = xp + yp;

 printf("Result = %d", result);
     return 0;
 }
```

Notice that in the example above we have to write an expression like *variable * variable * variable* each time we have to calculate a cube. In addition to the fact that we have to repeat common patterns throughout our code, programs that are written in this form force us to work always at the level of the particular operations that happen to be primitives in the language (multiplication, in this case) rather than in terms of higher-level operations (taking a cube in this case). Our programs would be able to compute cubes, but our language would lack the ability to express the concept of cubing. One of the things we should demand from a powerful programming language is the ability to build abstractions by assigning names to common patterns and then to work in terms of the abstractions directly. *Functions* provide this ability. This is why all but the most primitive programming languages include mechanisms for defining Functions. The common pattern in our example program is that of multiplying a variable three time by itself. Let us see how we can define the concept of a cube using a function. Here is the same program as above but this time we will use a function.

Header

```
#include <stdio.h>
```

function declaration for `intcube()`

```
long intcube(int);
```

function definition for `main()`

```
int main(void)
{
    int x,y;
long xp, yp, result;

    printf("Enter the values for x and y");
    scanf("%d %d", &x, &y);

    xp = intcube(x);
    yp = intcube(y);

    result = xp + yp;

printf("Result = %d", result);
    return 0;
}
```

function definition for `intcube()`

```
long intcube(int n)
{
    Long n3;
    n3 = n * n * n;
    return n3;
}
```

fuction header

fuction body

Analysis Of the Program

In this program we have used and defined a function called `intcube()`. Whenever we want to calculate cube of the value in any integer variable, say x, we use the expression `intcube(x)`. The compiler know that certain instructions are to be used to calculate expressions like `intcube(x)`, and it places this information in the compiled program. When we run the program those instructions are run and their output is used in place of the expression `intcube(x)`. How does the compiler know which instructions are to be used to evaluate these expressions? It looks for the definition of the function `intcube()`.

19.2 Function Definition

The C code that describes what a function does is called the *function definition*. Function definition is different from *function declaration*, which is described in the section 9.4. As you can see the function definition for `intcube()`has been put at the end of the program. A function definition has the following general form

data type function_name(parameter list)
{
    declarations
    statements
}

A *function definition* can be divided into two parts. Every thing before the first brace is called the *header* of the function definition, and everything between the braces makes up the *body* of the function definition. In computer science using expressions like `intcube(x)` is know as *calling a function*, and the expression itself is called a *function call*. Whenever we call a function like `intcube(x)`, a certain group of instruction are executed. We can instruct the computer to use the value of certain variables in these instructions. This done by passing the value of these variables as *parameters* to the function. Parameters are placed between the parentheses `( )` in the function call. For example in the `intcube()` function, when we want the instructions to use the value of variable x in the calculation we use the expression `intcube(x)`, and when we want the same set of instructions to use the value of the variable y we use the expression `intcube(y)`. In the function definition we have to tell the compiler the number and data type of the values that will be passed as parameters to the function. This is done using the *parameter list*. Since we are only passing a single value of type int to the function (the number which is to be cubed), the function definition just has one value in the parameter list `(int n)` . When the instructions are run they may produce a result. When we use the return statement in our function definition, the value of this result is assigned to the value of the expression that called the function, say `intcube(y)`. We also have to tell the compiler what kind of value we are going to return. This is done by placing the *data type* of the return value before the function name in the function definition. Our `intcube()` function returns value of type `long`, therefore the cube function has `long` as its data type.

19.3 Function Call Process

Let us go over the process of a function call step by step using our `intcube()` function as an example.

1. We want to calculate the cube of the value in variable x, so we make a function call in our program by using the expression `intcube(x)` in the statement

   ```
   xp = intcube(x);
   ```

2. The program passes the value of x to the function `intcube()`. It does this by assigning the value of variable x to the variable n of the cube function. The value

of the variable n will now be used in the instructions of the function.

3. Notice that the next instruction to be executed in not going to be

   In other words the control has been passed to the function `intcube()`.

4. To calculate the cube the computer executes the instruction

   defined in the function definition. The value of n3 is now returned to the calling statement by using the statement

5. The control now passes back to the main program and the value of returned by the function is assigned to the variable xp. The next instruction to be executed is

19.4 Function Declarations

Functions should be declared before they are used. A function declaration the compiler the number and type of arguments that are to be passed to the function and the type of the value that is to be returned by the function. For example in our program above we have the declaration

This tells the compiler that `intcube()` is a function that takes a single argument of type int and returns a long. The general form of a function declaration is

type function_name (parameter type list)

If no value is returned by the function, the keyword void is used as its type.

19.5 Advantages of Using Functions

- Functions improve the readability of a program. A set of operations that is needed in several places in a program are replace by a single call to a function.
- Programs are easier to maintain and modify if we use functions. For example if we have to modify what a function does, only the code in the function needs to be changed, rest of the program remains the same.
- We know that solving a problem becomes easier if we can break it up into a set of smaller tasks. Similarly writing a large program becomes easier if we can divide it into smaller parts each of which takes the form of a function.

19.6 C Program Structure and the `main()` function

In chapter 2 we told you that all C programs begin execution with `main()`. It is important to realize that `main()` is also a function. A C program consists of one or more functions. Functions are defined as individual and cannot be nested. In our example in this chapter we defined the main function first and then after that we defined the function `intcube()`. Program execution always begins with the `main()` function. The `main()` function can then call other functions such as `intcube()`. Some of the functions have already been defined in a C system. These functions are called library functions. We have already used library functions `printf()` and `scanf()` in our programs. The declarations for `printf()` and `scanf()` are in the `stdio.h` file that is why we include it in our program.

Exercises

1. Write a function **double power(double x, int n)** that calculates x^n.
2. Write a function **int compare(int a, int b)** that compares two numbers a and b and returns a value 1 if a > b else it returns a value of 0.
3. Can you guess what the following program prints? Run it and compare the output with your answer. If you cannot explain the results ask you teacher.

```
  #include <stdio.h>

int z;

void f(int x)
{
    x=2;
    z += x;
}

int main(void)
{
   z = 5;
   f(z);
   printf("”, z);
   return 0;
}
```

All rights reserved with the
Punjab Textbook Board, Lahore.

4307

Printed in	Exp.Edition	Impression	Copies	Price
June 2002	Ist	2nd	15000	39.25